A Knowledge-Based Compositional Proof System for Parallel Processes

M. van Hulst and J.-J.Ch. Meyer

UU-CS-1996-19
May 1996

ISSN: 0924-3275

A Knowledge-Based Compositional Proof System for Parallel Processes

M. van Hulst J.-J.Ch. Meyer

Abstract

We show how epistemic logic may be used to reason about concurrent programs. Starting out from Halpern & Moses' interpretation of knowledge in the context of distributed systems, where they use the interleaving model, we extend this to a setting where also truly concurrent computations can be modeled, viz. posets of action labels. Moreover, and more importantly, we present an epistemic proof system for the the compositional verification of concurrent programs. As our programming language, we fix a channeled variant of Hoare's well-known Concurrent Sequential Processes (CSP). Proofs of soundness and (relative) completeness of the proof system are provided.

1 Introduction

In [HM85] Halpern and Moses presented a framework to reason about distributed processes based on the notion of knowledge. They showed that a modal logic of knowledge (epistemic logic) may be employed fruitfully to reason about the behaviour of networks of processors in which communication protocols take care of the flow of messages between the processors. They were able to prove a number of fundamental results regarding the kind of knowledge that is or is not reachable in such networks ([HM90]). Moreover, using epistemic logic it appeared to be possible to prove the correctness of some well-known protocols such as the alternating bit protocol ([HZ87]).

The question we address in this paper is whether epistemic logic is useful as well in the verification and specification of parallel or concurrent programs in a compositional framework. To get a feel for the idea of using epistemic notions (i.e., notions pertaining to knowledge) consider a command $P_i?x$, expressing a request for a value from process i to be stored in the variable x. Restricting ourselves to synchronous communication, of course such a command will only be executed successfully when it is used in a process j, if process i — that is executed in parallel with j — is willing to send a value to process j, and execution of process j will be suspended until this happens. Possibly it will never happen, and then the process j will fail to be executed. But suppose that a successful transmission of a value takes place from the process i to the process j. Then it is not known a priori to the process j which value it will receive to store in x (possibly apart from some information regarding the type of the variable x, e.g. integers), since this depends entirely on process i. The process j must consider possible all values that are allowed by the type of x.

In the 'classical' (Hoare logic-based) proof system for CSP ([AFdR80]) this local uncertainty or rather ignorance is represented by an axiom $\{p\}P_i?x\{q\}$, meaning that if p holds before execution of the command $P_i?x$, then q holds after execution (provided that the execution is successful), for arbitrary p and q. In effect this means that it is left completely open what happens after execution of $P_i?x$! The approach of Apt et al. corrects this arbitrariness by enforcing a co-operation test between the local proofs of the correctness of the (sequential) processes involved. So, the idea amounts to give (guess) correctness proofs of the sequential processes, after which these proofs are checked on global consistency by the co-operation test.

In the approach we propose in this paper we shall directly use notions of knowledge to express the uncertainty in cases as above. More in particular, we employ modal operators K_i to express that something is known to process i. We believe that in proceeding in this way we obtain a natural form of compositionality in the verification of concurrent programs, since thus the language is enriched with pointers to the local processes, which may be used to speak only of the facts that are known locally to some process in isolation. This knowledge may later be combined to reason about a composite process, e.g. the request for receiving a message and the matching request for sending one in parallel. In fact, in 'standard' proof systems for the correctness of nondeterministic programs often a notion of knowledge is left implicit. In our approach this is made explicit by the employment of epistemic operators.

Semantically speaking, the proof system to be presented essentially consists of two layers, one dealing with the usual reasoning about a process in isolation, and the other dealing with programs possibly consisting of more programs in parallel. The first layer concerns the realm of set semantics, the second concerns that of Kripke-semantics.

To get an idea of our approach, we focus again on the command $P_i?x$ occurring in the process j. Although it is not known to j what is the value of x after its execution, which we may now express by the fact that, for any value v, the formula $K_j(x = v)$ does not hold, we *do* know that if the execution has been successfully completed it is the case that x has some value in the domain of the variable x and that this value has been sent from process i to the process j. This may be expressed by something like $K_j(\exists v[x = v \wedge sent(v, i, j)])$. In fact, as we shall see we even know some more, viz. concerning the locations of the commands of sending and receiving, respectively, in the processes i and j. We shall postpone the discussion of this to the formal treatment in the sequel.

The choices made in this paper represent but one from a vast number of possibilities. In particular, our choice to consider CSP is a first test how things work out in a concrete and established setting. In section 11 it is shown how the method can be easily adapted to cope with asynchronous communication as well. We envisage to investigate extensions of our framework in at least two directions: firstly, considering more complex programming languages, such as e.g. (subsets of) POOL ([Ame89]) in order to see the practical use of epistemic notions in the context of more advanced programming languages, particularly object-oriented ones. These objects act as natural agents to which epistemic operators may refer. Secondly, we could consider extensions of the logic with more expressive power, such as the incorporation of temporal operators. This may also be interesting in itself for devising a logic to reason about truly concurrent processes.

Compared to the work of Halpern and Moses, our notion of knowledge is somewhat different in the sense that we use knowledge as a meta-notion, in order to prove properties of programs in a compositional way. Halpern and Moses are not interested in compositionality of programs but rather use programming constructs which involve tests on the knowledge of the processes (agents) in a program. Programs which have this property are then called 'knowledge-based protocols'.

2 Syntax of the Programming Language

We will now give the syntax of our programming language, which is a variant of CSP ([Hoa78]). Assume a set CHAN of communication channels, and a set VAR of variables, both finite. We will denote the variables of a statement S (or program for that matter) by VAR(S) respectively. Labels in programs are denoted by l, l_i etcetera.

$$\begin{array}{lrl}
\textit{Bexpression} & b ::= & e = e' \mid e < e' \mid \neg b \mid b \wedge b' \\
\textit{Expression} & e ::= & v \mid x \mid e + e' \mid e - e' \mid e \times e' \\
\textit{Basic command} & s ::= & \textit{skip} \mid x := e \mid c!e \mid c?x \\
\textit{Statement} & S ::= & l : s \mid S; S' \mid [\![_{i=1}^{m} [b_i; l_i : c_i?x_i \rightarrow S_i] \mid \\
 & & \star[\![_{i=1}^{m} [b_i; l_i : c_i?x_i \rightarrow S_i] \\
\textit{Program} & PR ::= & [P_1 :: S_1 \parallel \cdots \parallel P_n :: S_n]
\end{array}$$

Furthermore, we have the following *syntactical restrictions*:

- A label can occur only once in a program PR
- For $S_i, S_j \in PR, i \neq j$ it holds that VAR(S_i) $\cap$ VAR(S_j) $= \emptyset$
- For $S; S'$: if S contains $c!e(c?x)$ then S' does not contain $c?x(c!e)$
- For $(\star)[\![_{i=1}^{m} [b_i; l_i : c_i?x_i \rightarrow S_i]$: for all $i, j \in \{1, ..., m\}$, S_j does not contain $c_i!e$, and if S_i contains $c!e$ then S_j does not contain $c?x$.
- For $P_1 :: S_1 \parallel \cdots \parallel P_n :: S_n \in PR$: if $c?x(c!e) \in S_i$ then not $c?x'(c!e') \in S_j$, where $i, j \in \{1, \ldots, n\}, i \neq j$

These syntactic restrictions guarantee that every variable occurring in a program is local to some particular process; moreover, each channel in CHAN is unidirectional and connected to at most 2 processes. Furthermore, we assume programs are closed, which means that each channel is connected to *exactly* two processes.

In this syntax, v is a constant, *skip* is the null command, $x := e$ denotes an assignment, $c!e$ denotes 'output to process i the value of e on channel c', and $c?x$ denotes 'input a value from channel c and store it in x'. Typically, basic commands will be denoted by s. As is the case in CSP, communication will be modeled synchronously; a process willing to send some value over a channel has to wait (becomes blocked) until the corresponding receiving process executes the corresponding receive command, and vice versa. In section 11 we will indicate how adaptations can be made in order to support asynchronous communication.

Furthermore, a statement S is a labeled basic command, and the operation ; denotes sequential composition. The execution of a (guarded) statement $[\![_{i=1}^{m} [b_i; l_i : c_i?x_i \rightarrow S_i]$ selects one of the alternatives for which the boolean guard is true and the corresponding process is willing to send a value; the communication is executed and control is passed to the statement S_i from the selected branch. In case none of the booleans evaluates to true the guarded statement terminates. The execution of the recursive guarded statement $\star\|_{i=1}^{m}[b_i; l_i : c_i?x_i \rightarrow S_i]$ consists of executing the guarded statement until all booleans evaluate to false.

A program PR finally is the parallel composition of a number of processes, where a process is a statement labeled with an —indexed— process label P.

3 Semantics of the Programming Language

In this section, we prepare grounds for the use of first-order epistemic logic, of which our assertion language —to be defined in Section 4— is an instance. As usual, we use Kripke models to interpret our logic.

In order to do so, we first define the *view* semantics for individual processes, which consists of a set semantics. Then, we proceed to define the Kripke semantics of programs, thereby using the view semantics of an individual process to define the reachability (possible worlds) relation of that process. We then are able to interpret Hoare-triples containing (also) epistemic assertions, to be defined later on.

The semantics to be defined below will be aimed at describing the changes in the valuation of the program variables and recording the communicated values. This is sufficient for our present purpose, namely the definition of a proof system for partial correctness. In order to prove more evolving properties such as deadlock freedom and progress properties, the semantics would have to be enhanced to convey more information. Not to blur our presentation, which is aimed at introducing epistemic logic in the assertion language, we decided to stick to partial correctness. There seem to be no real obstacles for the above-mentioned enhancements, however.

3.1 The Semantical Domain

In the following, let PR be a program, consisting of n processes. Let $\mathcal{P}$ denote the set $\{1, ..., n\}$ of process-indices in PR. We define $\mathcal{S}$ with typical element σ to be the set of valuations of $\mathsf{VAR}(PR) \bigcup \mathsf{LVAR}$, where LVAR with typical elements x (hx) is the set of logical (history) variables. σ maps elements of VAR and logical variables onto the domain $\mathbb{Z}$, with typical element v, booleans onto the domain $\{\mathbf{true}, \mathbf{false}\}$, and logical history variables hx onto $\mathcal{H}$, to be defined shortly.

We will use the notation $\sigma[v/x]$ to denote the valuation function which is equal to σ but for the valuation of x, which is v.

Next we define the set of program labels:

$\Lambda = \{l, \langle v, c, ?, m\rangle, \langle v, c, l, ?\rangle, \langle v, c, l, m\rangle \mid l, m \text{ appear in } PR\}$

Thus, there are two possible formats for labels, with as intended meaning that a 'simple' label l reflects some internal action, whereas a 'quadruple' label describes a communication, or an attempt at a communication. For instance, the label $\langle v, c, l, m\rangle$ informally describes the sending of value v over channel c where the sending statement is labeled l and the receiving statement is labeled m. The appearance of the question marks in the quadruple labels is due to incomplete information: they typically occur in the semantics of communication statements in isolation. In the sequel, both simple and quadruple labels will be denoted by $\lambda, \mu, ...$ when we are not interested in their inner structure. The set of i-labels, Λ_i, consists of the simple labels from Λ which appear syntactically in P_i and the quadruple labels from Λ which contain both a question mark and a simple label which appears syntactically in P_i. The set of global labels, Λ_0, consists of the labels from Λ that do not contain a question mark.

Finally, we define the set of histories $\mathcal{H}$, with typical element h, as the set of posets $(H, <)$ over Λ_0 with the restriction that for any i, $H \cap \Lambda_i$ is totally ordered by $<$. As will be explained below, the notion of poset is adequate, as we (implicitly) generate different labels when encountering the same label more than once in the event of iteration.

The basic building blocks of our semantical domain, points, are pairs $\langle \sigma, h \rangle \in \mathcal{S} \times \mathcal{H}$, where the first component of a pair is a state, and the second component describes the history via which the state in the first component was reached. We sometimes abuse notation and write $\lambda \in h$ when we mean $\lambda \in H$, where $h = (H, <)$. Furthermore, when h is a poset with only one element (hence H is a singleton), we may use that element as denotation for h, and ϵ denotes the empty history.

In the next section, individual processes will be provided with a semantics, which we will call *view* semantics, because of the fact that it provides local processes with certain, limited information of the overall program behaviour. Therefore, next to the full domain we will use a *local* domain for each process P_i: $\mathcal{S}_i \times \mathcal{H}_i$, where $\mathcal{S}_i$ is defined analogously to $\mathcal{S}$ but for the fact that valuations are now restricted to variables of process i and $\mathcal{H}_i$ are linear posets (sequences) over Λ_i. Analogous to the global case, elements from $\mathcal{S}_i \times \mathcal{H}_i$ will be called *local points*. We will use the dot "$\cdot$" to denote the concatenation operation between (local) histories. Because in our framework the concatenation operator will only be needed in determining the local semantics of processes, and hence only for the concatenation of linear posets, its meaning boils down to the usual concatenation of sequences.

3.2 View Semantics for Statements

In this section we will provide the semantic clauses giving meaning (assigning views) to statements. Assume in the following that S is a statement of process i. The semantical operator, which transforms sets of local points to sets of local points, is typed as follows:

$$[\![\cdot]\!] : S \times \wp(\mathcal{S}_i \times \mathcal{H}_i) \rightarrow \wp(\mathcal{S}_i \times \mathcal{H}_i)$$

It is defined point-wise, as in e.g. [FLP84], which means we only have to define $[\![S]\!](\langle \sigma, h \rangle)$, for all pairs $\langle \sigma, h \rangle$. Once this is done, we derive the semantics for sets of points as follows: $[\![S]\!](V) = \bigcup_{\langle \sigma, h \rangle \in V}([\![S]\!](\langle \sigma, h \rangle))$. Note that we use $\langle \sigma, h \rangle$ also to denote elements of $\mathcal{S}_i \times \mathcal{H}_i$.

For the skip-statement, the semantics is simple: the valuation function remains the same, while the history is augmented with the label of the statement. Note that $h \cdot l$ is a sequence concatenation.

•$[\![l : skip]\!](\langle \sigma, h \rangle) = \{\langle \sigma, h \cdot l \rangle\}$

Define e_σ (to be denoted by $\sigma(e)$ also) as the value of e in σ.

•$[\![l : x := e]\!](\langle \sigma, h \rangle) = \{\langle \sigma[e_\sigma/x], h \cdot l \rangle\}$

The semantics of the output-statement is obtained by augmenting the history with a quadruple label which expresses that a corresponding communication has taken place. The question mark within the quadruple label indicates that the label of the receiving statement is not available locally.

$\bullet[\![l : c!e]\!](\langle\sigma, h\rangle) = \{\langle\sigma, h \cdot \langle e_\sigma, c, l, ?\rangle\rangle\}$

Regarding the input-statement, all pairs of changed states and extended histories describing possible communications are included:

$\bullet[\![l : c?x]\!](\langle\sigma, h\rangle) = \{\langle\sigma[v/x], h \cdot \langle v, c, ?, l\rangle\rangle \mid v \in \mathbb{Z}\}$

The semantic clause for sequential composition:

$\bullet[\![S_1; S_2]\!](\langle\sigma, h\rangle) = [\![S_2]\!]([\![S_1]\!](\langle\sigma, h\rangle))$

The semantic clause for the nondeterministic statement:

$\bullet[\![\|_{i=1}^{m}[b_i; l_i : c_i?x_i \rightarrow S_i]]\!](\langle\sigma, h\rangle) = \{\langle\sigma, h\rangle \mid \sigma(b_i) = \textbf{false}, \text{ all } i\} \cup \{\langle\sigma', h'\rangle \mid$ there exists $k \leq m$ such that $\sigma(b_k) = \textbf{true}$ and $\langle\sigma', h'\rangle \in [\![l_k : c_k?x_k; S_k]\!](\langle\sigma, h\rangle)\}$

Finally, the clause for the iterated guarded command:

$\bullet[\![\star\|_{i=1}^{m}[b_i; l_i : c_i?x_i \rightarrow S_i]]\!](\langle\sigma, h\rangle) = \{\langle\sigma', h'\rangle \mid$ there exists $k, \langle\sigma_1, h_1\rangle, ..., \langle\sigma_k, h_k\rangle$ such that $\sigma' = \sigma_k$, $h' = h_k$, $\sigma_k(b_i) = \textbf{false}$ for all i and for all $0 \leq j < k$: for some i, $\sigma_j(b_i) = \textbf{true}$ and $\langle\sigma_{j+1}, h_{j+1}\rangle \in [\![l_i : c_i?x_i; S_i]\!](\langle\sigma_j, h_j\rangle)\}$ (where $\sigma_0 \overset{\text{def}}{=} \sigma$ and $h_0 \overset{\text{def}}{=} h$)

Although according to this last clause it is possible to obtain histories that contain multiple occurrences of the same label, it is not difficult to circumvent this by means of a counting mechanism. Formally this can be accomplished by keeping track in σ for every label l the number of times it has been used, and for instance pairing l with this number, thus obtaining unique labels. To avoid cumbersome notation, we leave this semantical feature hidden.

3.3 Kripke-style Semantics of Programs

Now that we have defined the view semantics of statements, we are able to define the semantics of programs, possibly composed of several processes. Our domain will be a pair consisting of a Kripke structure $\mathcal{M} = (\mathcal{S} \times \mathcal{H}, R_1, ..., R_n)$ and a pair $\langle\sigma, h\rangle \in \mathcal{S} \times \mathcal{H}$, where the relation R_i represents the accessibility relation of process i, giving the points that are "equivalent" as far as process i is concerned. In most approaches, a Kripke model additionally contains a truth-function to atomic propositions per state. In our case, its role would be quite insignificant, because the worlds themselves convey the necessary information through the state-functions.

Any formula in an epistemic logic is validated with respect to both a model $\mathcal{M}$ and a world s. The pair $\langle\sigma, h\rangle$ fulfils this role of current world in the Kripke semantics of statements and assertions.

A few words are in order here to explain the emerging Kripke structure, in particular concerning the relations R_i. Each equivalence relation R_i, belonging to process i, is derived from the semantics V_i of the statement that process i contains, in a simple fashion: V_i divides the set $\mathcal{S} \times \mathcal{H}$ into two classes in a trivial way. In a first attempt, the resulting relation, which is written R_{V_i} can be characterised as follows (the formal definition of the restriction operator $\restriction$ follows; note that this is a relation between *global* points):

$$\langle\sigma,h\rangle R_{V_i}\langle\sigma',h'\rangle \iff [\langle\sigma,h\rangle \restriction i \in V_i \Leftrightarrow \langle\sigma',h'\rangle \restriction i \in V_i]$$

That is, two points are equivalent according to i iff their projections onto process i are either both in V_i or both *not* in V_i. The intuition behind this is the inability of i to distinguish points that are "the same locally", i.e. with respect to i. We may think of the set of worlds as a set of possible global states of the system during some execution. The accessibility relation R_i may be interpreted as follows: $\langle\sigma,h\rangle R_{V_i}\langle\sigma',h'\rangle$ holds if the agent (the *process*) considers $\langle\sigma',h'\rangle$ to be a possible alternative, given the world $\langle\sigma,h\rangle$ (or given the agent's information 'in' $\langle\sigma,h\rangle$): the observable information in $\langle\sigma,h\rangle$ and $\langle\sigma',h'\rangle$ coincide for process i. Putting it differently, the relation R_i may be considered to be some kind of 'epistemic compatibility relation' between the worlds in the model. Or, putting it still somewhat differently: R_i holds between $\langle\sigma,h\rangle$ and $\langle\sigma',h'\rangle$ if the world $\langle\sigma',h'\rangle$ is a possible 'extension' of agent i's information about $\langle\sigma,h\rangle$. For instance, if the agent knows that his local variable x has the value 4, then he will only consider those worlds as possible alternatives in which this is the case, and, conversely, if he considers (given his view) two worlds as possible alternatives that do not agree on the value of x, then he does not exactly know this value.

This interpretation should be contrasted with the dynamic logic interpretation (see e.g. [Gol87]) and the temporal logic interpretation ([MP92, Krö87]) of the relations R_i. In the dynamic logic framework, $\langle\sigma,h\rangle R_i\langle\sigma',h'\rangle$ holds if $\langle\sigma',h'\rangle$ is a state which can be obtained by executing the 'program' i in state $\langle\sigma,h\rangle$ (hence, a relation is associated with any program/statement). In temporal logic, the situation is yet different: here we have only one relation R, modeling the relation between successive points in time. Comparing our epistemic interpretation with these two alternatives, we observe that in our case, the accessibility relations do not model 'progress' of any kind, but rather observationally equivalent alternatives of some point in the computation.

It should be noted that our notion of local indistinguishability deviates from the related notion in the approach of Halpern et al, in a crucial way. Because in their approach, a point is determined *within* a so-called run, that is, using the information of other processes which run in parallel, full information can be deduced from that run with respect to the program variables of some isolated process. Limiting the set of (global) runs to those that correspond with executions of the whole program under consideration, this approach leads to a useful notion of knowledge in their framework.

However, this method is not suited for a *compositional* approach, in which we want to describe the a priori knowledge of a particular process *regardless of its context*. Obviously, the price that has to be paid for this compositionality is that now much more points are to be considered as locally equivalent, resulting in bigger equivalence classes and hence weaker knowledge. Thus, contrary to the approach advocated by Halpern and Moses, it can now be the case that a process does not know the values of its own variables, for example after executing a nondeterministic statement, or an input statement. As a consequence of this, the notion of view that we define here differs from that of Halpern and Moses in that we cannot fix a set of atomic propositions that determines the view of a process; in other words, the dynamic nature of the current view function prohibits the selection of some fixed set of basic propositions by which it is determined.

It will be clear, that, under our interpretation of R_i, it seems natural to take this relation to be an equivalence: given agent i's information about $\langle\sigma, h\rangle$, the agent will consider $\langle\sigma, h\rangle$ to be a possible alternative (i.e., R_i is *reflexive*); if agent i finds world $\langle\sigma'', h''\rangle$ to be a possible extension of his information about some world $\langle\sigma', h'\rangle$ which, on its turn, is held to be a reasonable alternative for some world $\langle\sigma, h\rangle$ then $\langle\sigma'', h''\rangle$ will be considered a possible extension $\langle\sigma, h\rangle$, given the agent's information about $\langle\sigma, h\rangle$ (i.e., R_i is *transitive*); finally, if the agent finds $\langle\sigma', h'\rangle$ to be epistemically compatible with $\langle\sigma, h\rangle$, then this will also be the other way around (i.e., R_i will also be *symmetric*).

We now define formally the operations of restriction and chaotic closure on global pairs $\langle\sigma, h\rangle$. These operators are needed in order to identify the right set of (global) points that comply to the view of one or more processes.

Definition 3.1 The restriction operator $\restriction: (\mathcal{S}\times\mathcal{H})\times\mathcal{P} \to \bigcup_{i\in\mathcal{P}}(\mathcal{S}_i\times\mathcal{H}_i)$ is defined as follows:

$$\langle\sigma, h\rangle \restriction i = \langle\sigma \restriction_s i, h \restriction_h i\rangle$$

where $\restriction_h: \mathcal{H}\times\mathcal{P} \to \bigcup_{i\in\mathcal{P}}\mathcal{H}_i$ and $\restriction_s: \mathcal{S}\times\mathcal{P} \to \bigcup_{i\in\mathcal{P}}\mathcal{S}_i$ are defined by:

$$(H, <) \restriction_h i = (H', <')$$

where $H' = \{l \in H \mid l \text{ is an } i\text{-label}\}$
$\quad\bigcup\{(v, c, ?, m) \mid \exists l[(v, c, l, m) \in H] \wedge m \text{ is an } i\text{-label}\}$
$\quad\bigcup\{(v, c, m, ?) \mid \exists l[(v, c, m, l) \in H] \wedge m \text{ is an } i\text{-label}\}$,
and—note that by the fact that channels are one-one, the missing element from the quadruple label is uniquely determined by H, so that we can use the notation $O(v, c, ?, m)$ to denote the H-element (original) from which $(v, c, ?, m)$ was derived—

$<' = (< \bigcap H' \times H')$
$\quad\bigcup\{(\lambda, \mu) \in H' \times H' \mid \lambda < O(\mu) \wedge \lambda \text{ is simple }, \mu \text{ is quadruple }\}$
$\quad\bigcup\{(\lambda, \mu) \in H' \times H' \mid O(\lambda) < \mu \wedge \lambda \text{ is quadruple }, \mu \text{ is simple }\}$
$\quad\bigcup\{(\lambda, \mu) \in H' \times H' \mid O(\lambda) < O(\mu) \wedge \lambda, \mu \text{ are quadruple }\}$

$$\sigma \restriction_s i = \sigma'$$

where $\sigma'(hx) = \sigma(hx) \restriction_h i$
$\quad\sigma'(x) = \sigma(x), x \in \mathsf{VAR}(P_i) \bigcup \mathsf{LVAR}$
$\quad\quad = \text{undefined, otherwise}$

The restriction operator defined above is intended to cut out non-local information. It therefore has to throw out (valuations of) non-local variables, and, with regard to histories, it only keeps the local simple labels while adapting the local quadruple labels so as to omit the non-local label which is contained in them. Note that any quadruple label must contain two simple labels of different sort; this follows from the syntactic restrictions (a process cannot send to or receive from itself). The—somewhat awkward looking—definition of $<'$ is needed to ensure that the ordering relation $<$ is inherited correctly by the revised quadruple labels.

Definition 3.2 The chaotic closure operators $CC' : \mathcal{S}_i \times \mathcal{H}_i \to \mathcal{P}(\mathcal{S}\times\mathcal{H})$ and $CC : \mathcal{P}(\mathcal{S}_i \times \mathcal{H}_i) \to \mathcal{P}(\mathcal{S}\times\mathcal{H})$ are defined as follows:

$$CC'(\langle\sigma, h\rangle) = \{\langle\sigma', h'\rangle \mid \langle\sigma', h'\rangle \restriction i = \langle\sigma, h\rangle\}$$

$$CC(H) = \bigcup_{\langle\sigma,h\rangle \in H} CC'(\langle\sigma,h\rangle)$$

The chaotic closure operators are used to yield the set of all possible extensions of (a set of) local points. The definition shows why this type of operator is also referred to as "inverse projection". It comes up naturally in the description of a compositional semantics for parallel processes, see e.g. [Zwi88].

Remark From these definitions, it follows that the $R_{\{\langle\sigma,h\rangle\}}$-equivalence class of a point $\langle\sigma,h\rangle$ in $\mathcal{S}_i \times \mathcal{H}_i$ is equal to $CC'(\langle\sigma,h\rangle)$.

The set $CC'(\langle\sigma,h\rangle)$ as defined above, yields too many global points $\langle\sigma,h\rangle$. This is because the ordering on the elements of h is not fully determined by the projections only. An example may clarify this:

Example 3.3 Suppose a local point $\langle\sigma,h\rangle$ with $h = \langle 3,c,?,l_1\rangle \cdot l_2$. Then, the following two global histories, together with an arbitrary extension of σ to a global state, determine two points that are within $CC'(\langle\sigma,h\rangle)$ (where we denote the partial order by arrows):

$$h_1 = \langle 3,c,m_1,l_1\rangle \rightarrow m_2 \rightarrow l_2$$

and

$$\begin{tikzcd}
 & m_2 \\
h_2 = \langle 3,c,m_1,l_1\rangle \arrow[ur] \arrow[dr] & \\
 & l_2
\end{tikzcd}$$

However, h_1 should be considered as an over-specification: it imposes an ordering on the labels m_2 and l_2 which are both simple labels of different processes, and thus need not be ordered.

In order to obtain the least restrictive points (i.e. those points that have the least restrictive histories) we will leave out of consideration points that are 'refinements' of other points.

We will therefore aim for the maximal parallel histories in $CC'(\langle\sigma,h\rangle)$, and dismiss all points that are not maximal. To do so, we need a refinement relation over histories:

Definition 3.4 The relation $\prec \subseteq \mathcal{H} \times \mathcal{H}$ is defined as follows. Let $h_1 = (H_1, <_1)$ and $h_2 = (H_2, <_2)$.

$$h_1 \prec h_2 \text{ iff } H_1 = H_2 \ \wedge\ <_1 \subset <_2$$

We will pronounce $h_1 \prec h_2$ also as 'h_2 refines h_1'.

It can easily be seen that in example 3.3 above, $h_2 \prec h_1$ holds. Finally, we define the abstraction function *maxpar*:

Definition 3.5 The abstraction function $maxpar : \mathcal{P}(\mathcal{H}) \to \mathcal{P}(\mathcal{H})$ is defined as follows:

$$maxpar(\mathsf{H}) = \{h \in \mathsf{H} \mid h \text{ is minimal w.r.t. } \prec\}$$

We can now formally define R_{V_i}, taking care not to include points which are not maximal:

Definition 3.6

$$\langle\sigma, h\rangle R_{V_i} \langle\sigma', h'\rangle \iff [\langle\sigma, h\rangle \in maxpar(CC(V_i)) \Leftrightarrow \langle\sigma', h'\rangle \in maxpar(CC(V_i))]$$

Note that R_{V_i} is a partial relation, defined on maximal parallel elements with respect to $CC(V_i)$ only.

Now we can define the function $POSS$ which serves to describe the set of worlds that are held possible by all the processes involved, starting from a common global point $\langle\sigma_0, h_0\rangle$.

Definition 3.7 The function $POSS$, which, given a point, describes the—global—possibilities according to the process(es) in $[P_1 :: S_1, ..., P_n :: S_n]$ is defined as follows:

$$POSS(P_i, \langle\sigma, h\rangle) = maxpar(CC([\![S_i]\!]_v(\langle\sigma, h\rangle \restriction i)))$$

$$POSS(P_1, ..., P_n, \langle\sigma, h\rangle) = maxpar(\bigcap_i CC([\![S_i]\!](\langle\sigma, h\rangle \restriction i)))$$

Another, more constructive definition (albeit in a slightly different context) can be found in [HHM93]. However, the intersection used in the definition of $POSS$ above allows us to relate the semantics with the logic in a clearer way.

Corollary 3.8

$$POSS(P_1, ..., P_n, \langle\sigma, h\rangle) = \bigcap_i POSS(P_i, \langle\sigma, h\rangle)$$

Proof direct from definition 3.7, and the fact that $maxpar$ distributes over intersection. $\square$

Finally, we have done enough preparatory work in order to provide the definition of the semantics for programs:

Definition 3.9 (semantics of programs) We define $\mathcal{K}$ as the set of all Kripke structures of the form $(\mathcal{S} \times \mathcal{H}, R_1, ..., R_n)$, for $n \in \mathbb{N}$. Now

$$[\![\cdot]\!] : PR \times (\mathcal{S} \times \mathcal{H}) \to \mathcal{K} \times (\mathcal{S} \times \mathcal{H})$$

is defined by

•$[\![[P_1 :: S_1 \parallel \cdots \parallel P_n :: S_n]]\!](\langle\sigma_0, h_0\rangle) = (\mathcal{M}, \langle\sigma, h\rangle)$, where

$$\begin{aligned}&\mathcal{M} = (\mathcal{S} \times \mathcal{H}, R_{V_1}, \ldots, R_{V_n}),\\ &V_i = [\![S_i]\!](\langle\sigma_0, h_0\rangle \upharpoonright i),\\ &\langle\sigma, h\rangle = f_c(POSS(P_1, \ldots, P_n, \langle\sigma_0, h_0\rangle)),\end{aligned}$$

where f_c is a choice function, picking an arbitrary pair $\langle\sigma, h\rangle$ from a given set in $\wp(\mathcal{S}\times\mathcal{H})$. The choice function is needed for technical reasons only, namely to obtain an arbitrary point out of this set, which, together with the model $\mathcal{M}$, constitutes a world.

In words, the Kripke model $[\![P_1 :: S_1 \parallel \cdots \parallel P_n :: S_n]\!](\langle\sigma_0, h_0\rangle)$ is obtained by first determining the view semantics V_i of the individual processes, and then lifting these to relations R_{V_i}. Furthermore the actual world $\langle\sigma, h\rangle$ has to be selected in such a way that it complies with the views of all processes.

It should be noted that, in our two-leveled approach, it is only after execution of a parallel program that we can say something sensible about the knowledge of one or more processes. Therefore, Kripke models only appear in the range of our semantical function, and not in the domain. More comments on this can be found in section 10.

4 Syntax of Formulae

In this section, we define our language of assertions. There will be three kinds of assertions: local assertions $Assn_i$, non-epistemic assertions $Assn_-$ and epistemic assertions $Assn$. These will correspond to the different correctness formulae to be defined further on. Moreover, we define sets of (local) expressions and history expressions:

$$\begin{array}{lrl}
Expr_i & e_i :: & v \mid x_i(\in \mathsf{VAR}_i) \mid e_i + e_i' \mid e_i - e_i' \mid e_i \times e_i' \\
Expr & e :: & v \mid x(\in \mathsf{VAR}) \mid e + e' \mid e - e' \mid e \times e' \\
Hexpr_i & he_i :: & \epsilon \mid l_i \mid \langle e_i, c, l_i, ?\rangle \mid \langle x_i, c, ?, l_i\rangle \mid he_i \cdot he_i' \mid hx \upharpoonright i \mid hist \upharpoonright i \\
Assn_i & \varphi_i :: & e_i = e_i' \mid he_i = he_i' \mid \neg\varphi_i \mid \varphi_i \wedge \varphi_i' \mid \exists x[\varphi_i] \mid \exists hx[\varphi_i] \\
Assn_- & \varphi_- :: & e = e' \mid \varphi_i \mid \varphi_- \wedge \varphi_-' \\
Assn & \varphi :: & \varphi_- \mid K_i\varphi \mid K_G\varphi
\end{array}$$

The definition of expressions is as in the definition of the syntax of the language. As to the history expressions, these consist of the empty history ϵ, or a (simple or quadruple) label, to be understood as a poset, or two history expressions composed sequentially, or the projection of a history variable hx, or the projection of the current history $hist$. Note that there are no global history expressions, forcing us to reason about history expressions on the local level only.

The local assertions consecutively denote equality of expressions, equality of history expressions, and the operations negation, conjunction and quantification. The non-epistemic assertions include equality of non-local expressions, the set of all local assertions and the operation of conjunction. The epistemic assertions finally consist of the non-epistemic assertions, and two predicates expressing the knowledge of a process of some local assertion, and the knowledge of a group of processes of some assertion.

In the sequel, we will use the expressions $hx = hx'$, $hx = hist$, etcetera to denote the assertions $\bigwedge_i hx \upharpoonright i = hx' \upharpoonright i$ and $\bigwedge_i hx \upharpoonright i = hist \upharpoonright i$.

5 Semantics of Formulae

Firstly, we need two valuation functions that assign a meaning to expressions and history expressions in a point.

Definition 5.1 The valuation function $\mathcal{V} : \mathit{Expr} \times (\mathcal{S} \times \mathcal{H}) \to \mathbb{Z}$ is defined as follows:

$$
\begin{aligned}
&\mathcal{V}(x)(\langle\sigma, h\rangle) = \sigma(x)\\
&\mathcal{V}(v)(\langle\sigma, h\rangle) = v\\
&\mathcal{V}(e_1 + e_2)(\langle\sigma, h\rangle) = \mathcal{V}(e_1)(\langle\sigma, h\rangle) + \mathcal{V}(e_2)(\langle\sigma, h\rangle)\\
&\mathcal{V}(e_1 - e_2)(\langle\sigma, h\rangle) = \mathcal{V}(e_1)(\langle\sigma, h\rangle) - \mathcal{V}(e_2)(\langle\sigma, h\rangle)\\
&\mathcal{V}(e_1 \times e_2)(\langle\sigma, h\rangle) = \mathcal{V}(e_1)(\langle\sigma, h\rangle) \times \mathcal{V}(e_2)(\langle\sigma, h\rangle)
\end{aligned}
$$

Similarly, the local valuation function $\mathcal{V}_i : \mathit{Expr}_i \times (\mathcal{S}_i \times \mathcal{H}_i) \to \mathbb{Z}$ is defined, replacing x by x_i in the first clause.

The valuation function $\mathcal{V}_{hi} : \mathit{Hexpr}_i \times (\mathcal{S}_i \times \mathcal{H}_i) \to \mathcal{H}_i$ is defined as follows:

$$
\begin{aligned}
&\mathcal{V}_{hi}(\epsilon)(\langle\sigma, h\rangle) = \epsilon\\
&\mathcal{V}_{hi}(l_i)(\langle\sigma, h\rangle) = l_i\\
&\mathcal{V}_{hi}(\langle e_i, c, l, ?\rangle)(\langle\sigma, h\rangle) = \langle \mathcal{V}_i(e_i)(\langle\sigma, h\rangle), c, l, ?\rangle\\
&\mathcal{V}_{hi}(\langle v, c, ?, l\rangle)(\langle\sigma, h\rangle) = \langle v, c, ?, l\rangle\\
&\mathcal{V}_{hi}(he_i \cdot he_i')(\langle\sigma, h\rangle) = \mathcal{V}_{hi}(he_i)(\langle\sigma, h\rangle) \cdot \mathcal{V}_{hi}(he_i')(\langle\sigma, h\rangle)\\
&\mathcal{V}_{hi}(hx \upharpoonright i)(\langle\sigma, h\rangle) = \sigma(hx)\\
&\mathcal{V}_{hi}(\mathit{hist} \upharpoonright i)(\langle\sigma, h\rangle) = h
\end{aligned}
$$

One would expect the functions $\mathcal{V}$ and $\mathcal{V}_i$ to yield the same results when given a local expression, and two corresponding points (i.e. a local point and a global point, where the local point is the projection of the global point with respect to i). This is formalised in the following lemma.

Lemma 5.2 *For all* $e_i \in \mathit{Expr}_i$*:*

$\mathcal{V}(e_i)(\langle\sigma, h\rangle) = \mathcal{V}_i(e_i)(\langle\sigma, h\rangle \upharpoonright i)$

Proof by a simple induction on e_i resp. he_i. □

Now we define three interpretation functions on assertions. The first function, $\mathcal{T}_i$ is used to interpret local assertions in local points from $\mathcal{S}_i \times \mathcal{H}_i$, the second function, $\mathcal{T}_-$ interprets non-epistemic assertions in global points from $\mathcal{S} \times \mathcal{H}$, and the third function $\mathcal{T}$ interprets assertions in a *world* consisting of a Kripke structure and a global point.

Definition 5.3 The function $\mathcal{T}_i : \mathit{Assn}_i \times (\mathcal{S}_i \times \mathcal{H}_i) \to \{\mathbf{true}, \mathbf{false}\}$ is defined as follows:

$$
\begin{aligned}
&\mathcal{T}_i(e_i = e_i')(\langle\sigma, h\rangle) = \mathcal{V}_i(e_i)(\langle\sigma, h\rangle) = \mathcal{V}_i(e')(\langle\sigma, h\rangle)\\
&\mathcal{T}_i(he_i = he_i')(\langle\sigma, h\rangle) = \mathcal{V}_{hi}(he_i)(\langle\sigma, h\rangle) = \mathcal{V}_{hi}(he_i')(\langle\sigma, h\rangle)\\
&\mathcal{T}_i(\neg\varphi_i)(\langle\sigma, h\rangle) = \text{not } \mathcal{T}_i(\varphi_i)(\langle\sigma, h\rangle)
\end{aligned}
$$

$$\mathcal{T}_i(\varphi_i \wedge \varphi_i')(\langle\sigma, h\rangle) = \mathcal{T}_i(\varphi_i)(\langle\sigma, h\rangle)\text{and } \mathcal{T}_i(\varphi_i')(\langle\sigma, h\rangle)$$
$$\mathcal{T}_i(\exists x[\varphi_i])(\langle\sigma, h\rangle) = \text{there exists } v \text{ such that } \mathcal{T}_i(\varphi_i)(\langle\sigma[v/x], h\rangle)$$
$$\mathcal{T}_i(\exists hx[\varphi_i])(\langle\sigma, h\rangle) = \text{there exists } h' \text{ such that } \mathcal{T}_i(\varphi_i)(\langle\sigma[h'/hx], h\rangle)$$

Definition 5.4 The function $\mathcal{T}_- : Assn_- \times (\mathcal{S} \times \mathcal{H}) \rightarrow \{\mathbf{true}, \mathbf{false}\}$ is defined as follows:

$$\mathcal{T}_-(e = e')(\langle\sigma, h\rangle) = \mathcal{V}(e)(\langle\sigma, h\rangle) = \mathcal{V}(e')(\langle\sigma, h\rangle)$$
$$\mathcal{T}_-(\varphi_i)(\langle\sigma, h\rangle) = \mathcal{T}_i(\varphi_i)(\langle\sigma, h\rangle \upharpoonright i)$$
$$\mathcal{T}_-(\varphi_- \wedge \varphi_-')(\langle\sigma, h\rangle) = \mathcal{T}_-(\varphi_-)(\langle\sigma, h\rangle) \text{ and } \mathcal{T}_-(\varphi_-')(\langle\sigma, h\rangle)$$

Definition 5.5 The function $\mathcal{T} : Assn \times \mathcal{K} \times (\mathcal{S} \times \mathcal{H}) \rightarrow \{\mathbf{true}, \mathbf{false}\}$ is defined as follows:

$$\mathcal{T}(\varphi_-)(\mathcal{M}, \langle\sigma, h\rangle) = \mathcal{T}_-(\varphi_-)(\langle\sigma, h\rangle)$$
$$\mathcal{T}(K_i\varphi)(\mathcal{M}, \langle\sigma, h\rangle) = \forall\langle\sigma', h'\rangle[\langle\sigma, h\rangle R_{V_i} \langle\sigma', h'\rangle \Rightarrow \mathcal{T}(\varphi)(\mathcal{M}, \langle\sigma', h'\rangle)]$$
$$\mathcal{T}(K_G\varphi)(\mathcal{M}, \langle\sigma, h\rangle) = \forall\langle\sigma', h'\rangle[\langle\sigma, h\rangle R_G \langle\sigma', h\rangle \Rightarrow \mathcal{T}(\varphi)(\mathcal{M}, \langle\sigma, h\rangle)]$$

where in the last clause, the relation R_G is defined by:

$$\langle\sigma, h\rangle R_G \langle\sigma', h'\rangle \Longleftrightarrow \forall i \in G[\langle\sigma, h\rangle R_{V_i} \langle\sigma', h'\rangle]$$

or, equivalently, $R_G = \bigcap_i R_{V_i}$.

Because, according to this last definition, a world is a possible world for the group G iff this is the case for all members of G, we obtain the desired behaviour of group knowledge: it is the knowledge that would result if the members of G somehow 'combined' their knowledge ([HM85]). For a recent description of group knowledge, which exposes some of its intricacies, see [vdHvLM95].

6 Reasoning about Programs

In order to define our correctness formulas further on, we need the definitions of validation as defined below. One should be aware of the limitations (with respect to the assertions that can be evaluated) of the semantical domains involved. For instance, in a local point, say from $\mathcal{S}_i \times \mathcal{H}_i$, only local assertions from $Assn_i$ can be evaluated; and in a global point from $\mathcal{S} \times \mathcal{H}$, only non-epistemic assertions (i.e. from $Assn_-$) can be evaluated. An epistemic assertion from $Assn$ can only be interpreted in a *world*.

Definition 6.1 For all φ_i, φ_-, and φ we define

$$(\mathcal{S}_i \times \mathcal{H}_i \ni)\langle\sigma, h\rangle \models_i \varphi_i \Leftrightarrow \mathcal{T}_i(\varphi_i)(\langle\sigma, h\rangle)$$

$$(\mathcal{S}_i \times \mathcal{H}_i \supseteq)V \models_i \varphi_i \Leftrightarrow \forall\langle\sigma, h\rangle \in V[\langle\sigma, h\rangle \models_i \varphi_i]$$

$$(\mathcal{S} \times \mathcal{H} \ni)\langle \sigma, h\rangle \models \varphi_- \Leftrightarrow \mathcal{T}_-(\varphi_-)(\langle \sigma, h\rangle)$$

$$(\mathcal{S} \times \mathcal{H} \supseteq)V \models \varphi_- \Leftrightarrow \forall \langle \sigma, h\rangle \in V[\langle \sigma, h\rangle \models \varphi_-]$$

$$(\mathcal{M}, \langle \sigma, h\rangle) \models \varphi \Leftrightarrow \mathcal{T}(\varphi)(\mathcal{M}, \langle \sigma, h\rangle)$$

We next proceed with what could be viewed as a generalisation of lemma 5.2. We want to express that evaluating a local i-assertion in a global point amounts to evaluating the assertion in the local point which is the i-restriction of that global point. Formally:

Lemma 6.2

$$\forall \varphi_i \in Assn_i \forall \langle \sigma, h\rangle \in \mathcal{S} \times \mathcal{H}[\langle \sigma, h\rangle \models \varphi_i \leftrightarrow \langle \sigma, h\rangle \restriction i \models_i \varphi_i]$$

Proof Direct from definition 5.4. □

We now come to the definition of two kinds of correctness formulae, connecting program semantics with semantics of assertions. These definitions provide the standard partial correctness interpretation of Hoare triples, in the context of our domain.

Definition 6.3

$$\models_i \{\varphi_i\}S_i\{\psi_i\} \Leftrightarrow \forall \langle \sigma, h\rangle \in \mathcal{S}_i \times \mathcal{H}_i[\langle \sigma, h\rangle \models_i \varphi_i \Rightarrow [\![S_i]\!](\langle \sigma, h\rangle) \models_i \psi_i]$$

$$\models \{\varphi_-\}PR\{\psi\} \Leftrightarrow \forall \mathcal{M} \in \mathcal{K} \forall \langle \sigma, h\rangle \in \mathcal{S} \times \mathcal{H}[\langle \sigma, h\rangle \models \varphi_- \Rightarrow [\![PR]\!](\mathcal{M}, \langle \sigma, h\rangle) \models \psi]$$

7 Proof System

The proof system is divided into three parts: a general part, a local part and a global part. The general part contains rules that hold in both the local and the global system. In the section on completeness below, we will show that, using our proof system, we can derive any valid formula (of a particular format), provided that we can derive all valid assertions in first-order arithmetic. This type of completeness is called *relative* completeness. For this reason, we import all first-order validities in the system. The local part contains rules and axioms to describe the individual program constructs of processes; the formulation of the axioms and rules in this part presuppose that the statements in them occur inside some process S_i. The global part deals with parallel constructs, or programs, and also covers knowledge-related issues. In the general part, the symbol S denotes either a process S or a program PR.

7.1 General Part

Axiom 1 *(tautologies)*

All valid assertions in first-order arithmetic

Axiom 2 *(K-axiom)*

$(K_i\varphi \wedge K_i(\varphi \rightarrow \psi)) \rightarrow K_i\psi$

Axiom 3 *(veridicality)*

$K_i\varphi \rightarrow \varphi$

Axiom 4 *(positive introspection)*

$K_i\varphi \rightarrow K_iK_i\varphi$

Axiom 5 *(negative introspection)*

$\neg K_i\varphi \rightarrow K_i\neg K_i\varphi$

Axiom $2'$–$5'$

replace i by G in axioms 2–5

Axiom 6 *(group knowledge)*

$K_i\varphi \rightarrow K_G\varphi,$ *where* $i \in G$

Rule 1 *(modus ponens)*

$$\frac{\varphi,\ \varphi \rightarrow \psi}{\psi}$$

Rule 2 *(necessitation)*

$$\frac{\varphi}{K_i\varphi}$$

Rule 3 *(generalisation)*

$$\frac{\varphi}{\forall x[\varphi]}$$

Rule 4 *(consequence)*

$$\frac{p \rightarrow p',\ \{p'\}\mathsf{S}\{q'\},\ q' \rightarrow q}{\{p\}\mathsf{S}\{q\}}$$

Rule 5 *(conjunction)*

$$\frac{\{p_1\}\mathsf{S}\{q_1\},\ \{p_2\}\mathsf{S}\{q_2\}}{\{p_1 \wedge p_2\}\mathsf{S}\{q_1 \wedge q_2\}}$$

Using the above rules, we can derive the following theorem, stating that conjunction distributes over (both individual and group) knowledge:

Lemma 7.1 *For all i we have*

$$\vdash K_i\varphi \wedge K_i\psi \leftrightarrow K_i(\varphi \wedge \psi).$$

and similarly with i replaced by G

7.2 Local Part

Axiom 7 *(skip)* $\{\varphi_i[hist \cdot l/hist]\}l : skip\{\varphi_i\}$

Axiom 8 *(assignment)* $\{\varphi_i[hist \cdot l/hist, e/x]\}l : x := e\{\varphi_i\}$

Axiom 9 *(output)* $\{\varphi_i[hist \cdot (e, c, l, ?)/hist]\}l : c!e\{\varphi_i\}$

Axiom 10 *(input)* $\{\forall x'[\varphi_i[hist \cdot (x', c, ?, l)/hist, x'/x]]\}l : c?x\{\varphi_i\}$

Rule 6 *(sequential composition)*

$$\frac{\{\varphi_i\}S_1\{\varphi_i''\}, \{\varphi_i''\}S_2\{\varphi_i'\}}{\{\varphi_i\}S_1; S_2\{\varphi_i'\}}$$

Rule 7 *(guarded statement)*

$$\frac{\{\varphi_i \wedge b_j\}l_j : c_j?x_j; S_j\{\varphi_i'\} \qquad \textit{for } 1 \leq j \leq m}{\{\varphi_i\}\|_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{(\varphi_i \wedge \bigwedge_{j=i}^{m} \neg b_j) \vee \varphi_i'\}}$$

Rule 8 *(recursive guarded statement)*

$$\frac{\{\varphi_i \wedge \bigvee_j b_j\}\|_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{\varphi_i\}}{\{\varphi_i\} \star \|_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{\varphi_i \wedge \bigwedge_j(\neg b_j)\}}$$

7.3 Global Part

Rule 9 *(K-introduction)* .

$$\frac{\{\varphi_i\}S_i\{\varphi_i'\}}{\{\varphi_i\}P_i :: S_i\{K_i\varphi_i'\}}$$

Rule 10 *(K-persistence)*

$$\frac{\{\varphi_-\}P_i :: S_i\{K_i\varphi_i\}(\textit{for } i = 1, \ldots, n)}{\{\varphi_-\}[P_1 :: S_1 \parallel \ldots \parallel P_n :: S_n]\{K_i\varphi_i\}}$$

Rule 11 *(variable substitution)* Let $he = hist$ or $he = hx' \neq hx$.

$$\frac{\{\varphi_-\}PR\{\psi\}}{\{\varphi_-[e/x, he/hx]\}PR\{\psi\}} \quad \textit{provided } x, hx \textit{ do not occur in } PR \textit{ or } \psi.$$

The variable substitution rule is used as usual to get rid of auxiliary variables.

It should be stated that for the notion of completeness that we consider, not all of the above axioms are needed. This can be seen in the proof of completeness in Section 9, in which the introspection axioms do not play any role. However they are sound in our setting, and we decided to include them to stress the fact that we are dealing with an $\mathcal{S}5$-logic.

7.4 Examples

1. Let $PR = [P_1 :: l : c?x \parallel P_2 :: m : c!5]$.
   $\vdash \{hist \lceil 1 = \epsilon\} l : c?x \{hist \lceil 1 = \langle x, c, ?, l\rangle\}$ (axiom 10)
   $\vdash \{hist \lceil 1 = \epsilon\} P_1 :: l : c?x \{K_1(hist \lceil 1 = \langle x, c, ?, l\rangle)\}$ (K-introduction)
   $\vdash \{hist \lceil 1 = \epsilon\} PR \{K_1(hist \lceil 1 = \langle x, c, ?, l\rangle)\}$ (K-persistence)
   $\vdash \{hist \lceil 2 = \epsilon\} m : c!5 \{hist \lceil 2 = \langle 5, c, m, ?\rangle\}$ (axiom 9)
   $\vdash \{hist \lceil 2 = \epsilon\} P_2 :: m : c!5 \{K_2(hist \lceil 2 = \langle 5, c, m, ?\rangle)\}$ (K-introduction)
   $\vdash \{hist \lceil 2 = \epsilon\} PR \{K_2(hist \lceil 2 = \langle 5, c, m, ?\rangle)\}$ (K-persistence)
   $\vdash \{hist \lceil 1 = \epsilon \wedge hist \lceil 2 = \epsilon\} PR \{K_1(hist \lceil 1 = \langle x, c, ?, l\rangle) \wedge$
   $K_2(hist \lceil 2 = \langle 5, c, m, ?\rangle)\}$ (Conjunction)
   $\vdash \{hist = \epsilon\} PR \{K_G(hist \lceil 1 = \langle x, c, ?, l\rangle) \wedge K_G(hist \lceil 2 = \langle 5, c, m, ?\rangle)\}$
   (Consequence, Group kn.; $G = \{1, 2\}$)
   $\vdash \{hist = \epsilon\} PR \{K_G(hist \lceil 1 = \langle x, c, ?, l\rangle \wedge hist \lceil 2 = \langle 5, c, m, ?\rangle)\}$
   (Consequence)
   $\vdash \{hist = \epsilon\} PR \{K_G(hist = \langle 5, c, m, l\rangle \wedge x = 5)\}$ (Consequence)

2. Let $PR = [P_1 :: S_1 \parallel P_2 :: S_2 \parallel P_3 :: S_3]$, where

   $S_1 \equiv l_{11} : c!0; l_{12} : c'!1$,
   $S_2 \equiv l_{21} : c'?x; l_{22} : c''!(x+1)$, and
   $S_3 \equiv (l_{31} : c?z; l_{32} : c''?y) [] (l_{33} : c''?y; l_{34} : c?z)$.

   Then we can derive in a similar way $\{true\} PR \{K_{\{1,2,3\}} x = 1 \wedge y = 2 \wedge z = 0\}$, and also $\{true\} PR \{K_{\{1,3\}} z = 0\}$ but not $\{true\} PR \{K_{\{2,3\}} y = 2\}$. Note that from this last fact it follows that the combined knowledge of processes 2 and 3 is not enough to derive the value of y. However, we can prove the formula $\{true\} PR \{\ K_{\{2,3\}} y = x + 1\}$. This example shows how in particular cases, we can derive useful knowledge within a subgroup of all processes involved. Moreover, information does not get 'lost' when considering larger groups of processes, as in traditional (non-epistemic) proof system for this kind of language.

8 Soundness

The axioms 1–7 and rules 1–3 together constitute the a sound and complete axiomatization of first order $\mathcal{S}5$ (see e.g. [HC84]). So in particular it follows from our semantics that these axioms and rules are sound.

The program axioms together with the rules for sequential composition and nondeterministic choice from the local part can be checked to be sound in a standard way, as is the case for the rules of consequence and conjunction.

There remains the proof of the K-introduction rule, the K-persistence rule and the variable substitution rule. As to the first, we even have the following stronger result

Proposition 8.1

$$\models_i \{\varphi_i\}S_i\{\psi_i\} \Leftrightarrow \models \{\varphi_i\}P_i :: S_i\{K_i\psi_i\}$$

Proof

"$\Rightarrow$" Suppose $\models_i \{\varphi_i\}S_i\{\psi_i\}$

$\Leftrightarrow \forall\langle\sigma,h\rangle \in \mathcal{S}_i \times \mathcal{H}_i[\langle\sigma,h\rangle \models_i \varphi_i \Rightarrow [\![S_i]\!]_v(\langle\sigma,h\rangle) \models_i \psi_i]$

To prove: $\models \{\varphi_i\}P_i :: S_i\{K_i\psi_i\}$

$\Leftrightarrow \forall\langle\sigma,h\rangle \in \mathcal{S} \times \mathcal{H}[\langle\sigma,h\rangle \models_i \varphi_i \Rightarrow [\![P_i :: S_i]\!](\langle\sigma,h\rangle) \models_i K_i\psi_i]$

So suppose $\langle\sigma,h\rangle \models \varphi_i$.

Then also $\langle\sigma,h\rangle \restriction i \models_i \varphi_i$, by Lemma 6.2

So, by assumption $[\![S_i]\!]_v(\langle\sigma,h\rangle \restriction i) \models_i \psi_i$

$\Leftrightarrow V_i \models_i \psi_i$, where $V_i = [\![S_i]\!]_v(\langle\sigma,h\rangle \restriction i)$

$\Leftrightarrow maxpar(CC(V_i)) \models \psi_i$ (again Lemma 6.2)

$\Leftrightarrow ((\mathcal{S} \times \mathcal{H},\pi,R_{V_i}), f_c(POSS(P_i,\langle\sigma,h\rangle))) \models K_i\psi_i$

$\Leftrightarrow [\![P_i :: S_i]\!](\langle\sigma,h\rangle) \models K_i\psi_i$

"$\Leftarrow$" Suppose $\models \{\varphi_i\}P_i :: S_i\{K_i\psi_i\}$.

To prove $\models_i \{\varphi_i\}S_i\{\psi_i\}$.

Suppose $\langle\sigma,h\rangle \models_i \varphi_i$, and define $V_i = [\![S_i]\!]_v(\langle\sigma,h\rangle)$.

Let $\langle\sigma',h'\rangle \in maxpar(CC(\langle\sigma,h\rangle))$, and thus $\langle\sigma',h'\rangle \models \varphi_i$.

By assumption $[\![P_i :: S_i]\!](\langle\sigma',h'\rangle) \models K_i\psi_i$

$\Leftrightarrow ((\mathcal{S} \times \mathcal{H},\pi,R_{V_i}), f_c(POSS(P_i,\langle\sigma,h\rangle)))) \models K_i\psi_i$

$\Leftrightarrow maxpar(CC(V_i)) \models \psi_i$

$\Leftrightarrow V_i \models_i \psi_i$, by Lemma 6.2

So $\models_i \{\varphi_i\}S_i\{\psi_i\}$

□

The soundness of the K-persistence rule is proven as follows:

Proof Let in the following $V_i = [\![S_i]\!]_v(\langle\sigma,h\rangle \restriction i)$, for all i.

Suppose $\models \{\varphi_-\}P_i :: S_i\{K_i\varphi_i\}$

$\Leftrightarrow \forall\langle\sigma,h\rangle[\langle\sigma,h\rangle \models \varphi_- \Rightarrow [\![P_i :: S_i]\!](\langle\sigma,h\rangle) \models K_i\varphi_i]$

$\Leftrightarrow \forall\langle\sigma,h\rangle[\langle\sigma,h\rangle \models \varphi_- \Rightarrow (\mathcal{S} \times \mathcal{H},\pi,R_{V_i}), f_c(POSS(P_i,\langle\sigma,h\rangle)) \models K_i\varphi_i]$

$\Leftrightarrow \forall\langle\sigma,h\rangle[\langle\sigma,h\rangle \models \varphi_- \Rightarrow maxpar(CC(V_i)) \models \varphi_i]$

$\Leftrightarrow \forall\langle\sigma,h\rangle[\langle\sigma,h\rangle \models \varphi_-$

$\quad \Rightarrow (\mathcal{S} \times \mathcal{H},\pi,R_{V_1},\ldots,R_{V_n}), f_c(POSS(P_1,\ldots,P_n,\langle\sigma,h\rangle)) \models K_i\varphi_i]$

$\Leftrightarrow \forall\langle\sigma,h\rangle[\langle\sigma,h\rangle \models \varphi_- \Rightarrow [\![P_1 :: S_1 \parallel \ldots \parallel P_n :: S_n]\!](\langle\sigma,h\rangle) \models K_i\varphi_i]$

$\Leftrightarrow \models \{\varphi_-\}[\![P_1 :: S_1 \parallel \ldots \parallel P_n :: S_n]\!]\{K_i\varphi_i\}$

□

Finally, we prove the soundness of the variable substitution rule:

Proof Let *he* be a meta-variable denoting either *hist* or some history variable in LHVAR which is different from hx.

Suppose $\langle\sigma,h\rangle \models \varphi_-[e/x, he/hx]$.

Define $\tilde{\sigma} = \sigma[\mathcal{V}(e)(\langle\sigma,h\rangle)/x, \sigma(he)/hx]$.

By lemma 9.4 below, $\langle\tilde{\sigma},h\rangle \models \varphi_-$.

Let $(\mathcal{M}', \langle\sigma', h'\rangle) = [\![PR]\!](\langle\sigma, h\rangle)$.
Define $\tilde{\sigma'} = \sigma'[\mathcal{V}(e)(\langle\sigma, h\rangle)/x, \sigma(he)/hx]$.
Since x, hx do not occur in PR, we have, for some suitable $\mathcal{M}''$:
$(\mathcal{M}'', \langle\tilde{\sigma'}, h'\rangle) = [\![PR]\!](\langle\tilde{\sigma}, h\rangle)$.
Then, by $\{\varphi_{-}\}PR\{\psi\}$, we have $(\mathcal{M}'', \langle\tilde{\sigma'}, h'\rangle) \models \psi$.
Since x, hx do not occur in ψ, we arrive at $(\mathcal{M}', \langle\sigma', h'\rangle) \models \psi$.
(note that $\mathcal{M}''$ and $\mathcal{M}'$ differ only with respect to the variables x, hx)

$\square$

9 Completeness

In order to prove relative completeness of our system (in fact, a particular, strongly related notion which we will call K-completeness), we extend $Assn_i$ by adding $sp(\varphi_i, S_i)$ to it, the strongest postcondition with respect to a statement and a local assertion. The semantics is given as follows, as an extension of definition 6.1:

$$(\mathcal{S}_i \times \mathcal{H}_i \ni)\langle\sigma, h\rangle \models_i sp(\varphi_i, S_i) \Leftrightarrow \exists\langle\sigma_0, h_0\rangle[\langle\sigma_0, h_0\rangle \models_i \varphi_i \wedge \langle\sigma, h\rangle \in [\![S_i]\!]_v(\langle\sigma_0, h_0\rangle)]$$

The proof that this definition indeed provides a semantical characterisation of the strongest postcondition is standard and not given here (see e.g.[dB80]).

We now have the following lemma stating that the strongest postcondition can be expressed in $Assn_i$, for any $\varphi_i \in Assn_i$ and statement S_i (for the recursive guarded statement, the situation is somewhat different, in that we need the existence of an invariant; see the proof of Theorem 9.6 below).

Lemma 9.1

- $\models_i sp(\varphi_i, l : skip) \leftrightarrow \exists hx : \varphi_i[hx/hist] \wedge hist \restriction i = hx \restriction i \cdot l$
- $\models_i sp(\varphi_i, l : x := e) \leftrightarrow \exists x', hx : \varphi_i[x'/x, hx/hist] \wedge x = e[x'/x] \wedge hist \restriction i = hx \restriction i \cdot l$
- $\models_i sp(\varphi_i, l : c!e) \leftrightarrow (\exists hx : \varphi_i[hx/hist] \wedge hist \restriction i = hx \restriction i \cdot (e, c, l, ?))$
- $\models_i sp(\varphi_i, l : c?x) \leftrightarrow (\exists x', hx : \varphi_i[x'/x, hx/hist] \wedge hist \restriction i = hx \restriction i \cdot (x, c, ?, l))$
- $\models_i sp(\varphi_i, S_1; S_2) \leftrightarrow sp(sp(\varphi_i, S_1), S_2)$
- $\models_i sp(\varphi_i, [\!]_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]) \leftrightarrow (\varphi_i \wedge \bigwedge_{j=1}^{m} \neg b_j) \wedge \bigvee_{j=1}^{m} sp(\varphi_i \wedge b_j, l_j : c_j?x_j; S_j)$

In order to prove this lemma, we first state some lemmas concerning substitution in expressions and assertions.

Assume the usual definition of substitution of variables in expressions and assertions; let $\varphi[x'/x]$ denote the formula φ where (logical) variable x is replaced by x'.

Lemma 9.2 $\mathcal{V}_i(e'_i[e_i/x])(\langle\sigma, h\rangle) = \mathcal{V}_i(e'_i)(\langle\sigma[\mathcal{V}_i(e_i)(\langle\sigma, h\rangle)/x], h\rangle)$

Lemma 9.3

- $\langle\sigma, h\rangle \models_i \varphi_i[e_i/x] \Leftrightarrow \langle\sigma[\mathcal{V}_i(e_i)(\langle\sigma, h\rangle)/x], h\rangle \models_i \varphi_i$

- $\langle\sigma,h\rangle \models_i \varphi_i[he_i/hx] \Leftrightarrow \langle\sigma[\mathcal{V}_{hi}(he_i)(\langle\sigma,h\rangle)/hx],h\rangle \models_i \varphi_i$
- $\langle\sigma,h\rangle \models_i \varphi_i[he_i/hist] \Leftrightarrow \langle\sigma,\mathcal{V}_{hi}(he_i)(\langle\sigma,h\rangle)\rangle \models_i \varphi_i$

Lemma 9.4 Let again he denote $hist$ or some $hx' \neq hx$.

- $\langle\sigma,h\rangle \models \varphi_-[e/x] \Leftrightarrow \langle\sigma[\mathcal{V}(e)(\langle\sigma,h\rangle)/x],h\rangle \models \varphi_-$
- $\langle\sigma,h\rangle \models \varphi_-[he/hx] \Leftrightarrow \langle\sigma[\sigma(he)/hx],h\rangle \models \varphi_-$

Proof (of lemma 9.1)

- $\langle\sigma,h\rangle \models sp(\varphi_i, l: skip)$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i$ and $\langle\sigma,h\rangle \in [\![l : skip]\!](\langle\sigma_0,h_0\rangle)]$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i$ and $\langle\sigma,h\rangle = \langle\sigma_0,h_0\cdot l\rangle)]$ iff
  $\exists h_0[\langle\sigma,h_0\rangle \models \varphi_i$ and $h = h_0\cdot l]$ iff
  $\exists h_0[\langle\sigma[h_0/hx],h_0\rangle \models \varphi_i$ and $h = h_0\cdot l]$ (hx fresh) iff
  $\exists h_0[\langle\sigma[h_0/hx],h\rangle \models \varphi_i[hx/hist]$ and $\langle\sigma[h_0/hx],h\rangle \models hist \upharpoonright i = hx \upharpoonright i\cdot l]$ iff
  $\exists h_0[\langle\sigma[h_0/hx],h\rangle \models \varphi_i[hx/hist] \wedge hist \upharpoonright i = hx \upharpoonright i\cdot l]$ iff
  $\langle\sigma,h\rangle \models \exists hx[\varphi_i[hx/hist] \wedge hist \upharpoonright i = hx \upharpoonright i\cdot l]$

- $\langle\sigma,h\rangle \models sp(\varphi_i, l: x := e)$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i$ and $\langle\sigma,h\rangle \in [\![l : x := e]\!](\langle\sigma_0,h_0\rangle)]$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i$ and $\sigma = \sigma_0[\sigma_0(e)/x], h = h_0\cdot l]$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0[\sigma_0(x)/x', h_0/hx],h\rangle \models \varphi_i[x'/x, hx/hist]$ and
  $\langle\sigma_0[\sigma_0(x)/x', h_0/hx],h\rangle \models x = e[x'/x]$ and $\sigma = \sigma_0[\sigma_0(e)/x], h = h_0\cdot l]$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma[\sigma_0(x)/x', h_0/hx],h\rangle \models \varphi_i[x'/x, hx/hist] \wedge x = e[x'/x] \wedge$
  $hist \upharpoonright i = hx \upharpoonright i\cdot l]$ iff
  $\langle\sigma,h\rangle \models \exists x', hx[\varphi_i[x'/x, hx/hist] \wedge x = e[x'/x] \wedge hist \upharpoonright i = hx \upharpoonright i\cdot l]$

- $\langle\sigma,h\rangle \models sp(\varphi_i, l: c!e)$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i$ and $\langle\sigma,h\rangle \in [\![l : c!e]\!](\langle\sigma_0,h_0\rangle)]$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i$ and $\langle\sigma,h\rangle = \langle\sigma_0,h_0\cdot\langle e_\sigma,c,l,?\rangle\rangle)]$ iff
  $\exists h_0[\langle\sigma,h_0\rangle \models \varphi_i$ and $h = h_0\cdot\langle e_\sigma,c,l,?\rangle]$ iff
  $\exists h_0[\langle\sigma[h_0/hx],h\rangle \models \varphi_i[hx/hist] \wedge hist \upharpoonright i = hx \upharpoonright i\cdot\langle e_\sigma,c,l,?\rangle]$ iff
  $\langle\sigma,h\rangle \models \exists hx[\varphi_i[hx/hist] \wedge hist \upharpoonright i = hx \upharpoonright i\cdot\langle e_\sigma,c,l,?\rangle]$

- $\langle\sigma,h\rangle \models sp(\varphi_i, l: c?x)$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i$ and $\langle\sigma,h\rangle \in [\![l : c?x]\!](\langle\sigma_0,h_0\rangle)]$ iff
  $\exists\langle\sigma_0,h_0\rangle, v[\langle\sigma_0,h_0\rangle \models \varphi_i$ and $\sigma = \sigma_0[v/x], h = h_0\cdot\langle v,c,?,l\rangle]$ iff
  $\exists\langle\sigma_0,h_0\rangle, v[\langle\sigma_0[\sigma_0(x)/x', h_0/hx],h\rangle \models \varphi_i[x'/x, hx/hist]$
  and $\sigma = \sigma_0[v/x], h = h_0\cdot\langle x_\sigma,c,?,l\rangle]$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma[\sigma_0(x)/x', h_0/hx],h\rangle \models \varphi_i[x'/x, hx/hist] \wedge$
  $hist \upharpoonright i = hx \upharpoonright i\cdot\langle x,c,?,l\rangle]$ iff
  $\langle\sigma,h\rangle \models \exists x', hx[\varphi_i[x'/x, hx/hist] \wedge hist \upharpoonright i = hx \upharpoonright i\cdot\langle x,c,?,l\rangle]$

- $\langle\sigma,h\rangle \models sp(\varphi_i, S_1; S_2)$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i \wedge \langle\sigma,h\rangle \in [\![S_1; S_2]\!](\langle\sigma_0,h_0\rangle)]$ iff
  $\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i \wedge \langle\sigma,h\rangle \in [\![S_2]\!]([\![S_1]\!](\langle\sigma_0,h_0\rangle))]$ iff
  $\exists\langle\sigma_0,h_0\rangle, \langle\sigma_1,h_1\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i \wedge \langle\sigma_1,h_1\rangle \in [\![S_1]\!](\langle\sigma_0,h_0\rangle) \wedge$
  $\langle\sigma,h\rangle \in [\![S_2]\!](\langle\sigma_1,h_1\rangle)]$ iff
  $\exists\langle\sigma_1,h_1\rangle[\exists\langle\sigma_0,h_0\rangle[\langle\sigma_0,h_0\rangle \models \varphi_i \wedge \langle\sigma_1,h_1\rangle \in [\![S_1]\!](\langle\sigma_0,h_0\rangle)] \wedge$
  $\langle\sigma,h\rangle \in [\![S_2]\!](\langle\sigma_1,h_1\rangle)]$ iff
  $\exists\langle\sigma_1,h_1\rangle[\langle\sigma_1,h_1\rangle \models sp(\varphi_i, S_1) \wedge \langle\sigma,h\rangle \in [\![S_2]\!](\langle\sigma_1,h_1\rangle)]$ iff
  $\langle\sigma,h\rangle \models sp(sp(\varphi_i, S_1), S_2)$

- $\langle\sigma, h\rangle \models sp(\varphi_i,]\!]_{j=1}^{m}[b_j; l_j : c_j?x_j \to S_j])$ iff
  $\exists\langle\sigma_0, h_0\rangle[\langle\sigma_0, h_0\rangle \models \varphi_i \wedge \langle\sigma, h\rangle \in [\![ ]\!]_{j=1}^{m}[b_j; l_j : c_j?x_j \to S_j]\!]((\langle\sigma_0, h_0\rangle))]$ iff
  $\exists\langle\sigma_0, h_0\rangle[\langle\sigma_0, h_0\rangle \models \varphi_i \wedge ((\langle\sigma_0, h_0\rangle = \langle\sigma, h\rangle \wedge \sigma(\bigwedge_j \neg b_j)\vee$
  $\exists k \leq m[\sigma_0(b_k) \wedge \langle\sigma, h\rangle \in [\![l_j : c_j?x_j; S_j]\!]((\langle\sigma_0, h_0\rangle)))]$ iff
  $\exists\langle\sigma_0, h_0\rangle[\langle\sigma_0, h_0\rangle \models \varphi_i \wedge \langle\sigma_0, h_0\rangle = \langle\sigma, h\rangle \wedge \sigma(\bigwedge_j \neg b_j)$ or
  $\exists\langle\sigma_0, h_0\rangle[\langle\sigma_0, h_0\rangle \models \varphi_i \wedge \exists k \leq m[\sigma_0(b_k)\wedge$
  $\langle\sigma, h\rangle \in [\![l_j : c_j?x_j; S_j]\!]((\langle\sigma_0, h_0\rangle)))]$ iff
  $\langle\sigma, h\rangle \models \varphi_i \wedge \bigwedge_j b_j$ or $\exists k \leq m[\langle\sigma, h\rangle \models sp(\varphi_i \wedge b_k, l_k : c_k?x_k; S_k)]$ iff
  $\langle\sigma, h\rangle \models \varphi_i \wedge \bigwedge_j b_j \vee \bigvee_{j=i}^{m} sp(\varphi_i \wedge b_j, l_j : c_j?x_j; S_j)$

□

The following lemma justifies the previously introduced syntactical abbreviation $hx = hist$ meaning $\bigwedge_i hx \upharpoonright i = hist \upharpoonright i$. Although in our assertion language there is no means of reasoning directly about global history expressions such as *hist*, it follows from this lemma that we *can* make global statements using the abbreviations.

Lemma 9.5 *A history $h \in \mathcal{H}$ is completely determined by all its projections $h \upharpoonright_h i$, where i ranges over all processes.*

Proof We show, for all $h, h' \in \mathcal{H}$:

$$h \neq h' \Rightarrow \exists i[h \upharpoonright_h i \neq h' \upharpoonright_h i] \qquad (*)$$

Let $h = (H, <)$ and $h' = (H', <')$, and suppose $h \neq h'$. We will use the following notation: $h \upharpoonright_h i = (H_i, <_i), h' \upharpoonright_h i = (H'_i, <'_i)$. As both h and h' are posets, there are the following possibilities:

- $H \neq H'$: without loss of generality, assume there is some i-label λ_i in $H \backslash H'$. There are the following possibilities:
  1. λ_i is a simple i-label: then also $\lambda_i \in H_i \backslash H'_i$, so $H_i \neq H'_i$, so $h \upharpoonright_h i \neq h' \upharpoonright_h i$.
  2. λ_i is a quadruple label, say $\langle v, c, l, m\rangle$, where l is an i-label and m is a j-label. Now $\langle v, c, l, ?\rangle \in H_i$ and $\langle v, c, ?, m\rangle \in H_j$, and it cannot be the case that $\langle v, c, l, ?\rangle \in H'_i$ and $\langle v, c, ?, m\rangle \in H'_j$ as will be shown in the rest of the proof. Suppose on the contrary that $\langle v, c, l, ?\rangle \in H'_i$ and $\langle v, c, ?, m\rangle \in H'_j$. Then $\langle v, c, l, m'\rangle \in H'$ and $\langle v, c, l', m\rangle \in H'$ for some l', m' and by unidirectedness of c, l' is an i-label and m' is a j-label. Furthermore, by the fact that $\langle v, c, l, m\rangle \in H \backslash H'$, we have $l \neq l', m \neq m'$. So H'_i contains at least the elements $\langle v, c, l, ?\rangle \in H_i$ and $\langle v, c, l', ?\rangle$, and H'_j contains at least the elements $\langle v, c, ?, m\rangle \in H_i$ and $\langle v, c, ?, m'\rangle$. Now if either $\langle v, c, l', ?\rangle \notin H_i$ or $\langle v, c, ?, m'\rangle \notin H_j$ then the consequence of $(*)$ holds so we are done. So suppose $\langle v, c, l', ?\rangle \in H_i$ and $\langle v, c, ?, m'\rangle \in H_j$. This implies $\langle v, c, l', m''\rangle \in H$ and $\langle v, c, l'', m'\rangle \in H$ for some l'', m''. Now suppose $m'' = m'$ (and so also $l'' = l'$, again by unidirectedness of c). Then there is an order conflict either between $h \upharpoonright_h i$ and $h' \upharpoonright_h i$ or between $h \upharpoonright_h j$ and $h' \upharpoonright_h j$: this is because the elements $\langle v, c, l, m\rangle$ and $\langle v, c, l', m'\rangle \in H$ must be ordered and also the elements $\langle v, c, l', m\rangle$ and $\langle v, c, l, m'\rangle \in H'$ must be ordered, and both orderings imply necessarily different orderings either on $h \upharpoonright_h i$ and $h' \upharpoonright_h i$ or on $h \upharpoonright_h j$ and $h' \upharpoonright_h j$.

So we conclude: $m'' \neq m'$ (otherwise we are done). Similarly we derive $l'' \neq l'$, so that we conclude: $\langle v, c, l'', m' \rangle$ and $\langle v, c, l', m'' \rangle \in H$, and thus $\langle v, c, l'', ? \rangle \in H_i$ and $\langle v, c, ?, m'' \rangle \in H_j$. Once again, if either $\langle v, c, l'', ? \rangle \notin H'_i$ or $\langle v, c, ?, m'' \rangle \notin H'_j$ then we are done as the consequence of $(*)$ holds. Analogously we conclude that there must be fresh m''', l''' etc., etc. So we can blow up both $h \lceil_h i$ and $h' \lceil_h i$ ad infinitum which contradicts the finiteness of our programs. So we conclude either $\langle v, c, l, ? \rangle \notin H'_i$ or $\langle v, c, ?, m \rangle \notin H'_j$, which leads to $h \lceil_h i \neq h' \lceil_h i$ or $h \lceil_h j \neq h' \lceil_h j$.

- $<\neq<'$ (while $H = H'$). Suppose $\lambda < \mu$ and $\lambda \not<' \mu$. Define $<_n$ ($<'_n$), the next-relation derived from $<$ ($<'$), by $\lambda <_n \mu \Leftrightarrow \lambda < \mu \wedge \neg\exists\nu[\lambda < \nu < \mu]$. Then there exists $k \in \mathbb{N} : \lambda <_n \lambda_1 <_n \lambda_2 <_n \ldots <_n \lambda_k <_n \mu$. It follows that for some m, $\lambda_m <_n \lambda_{m+1}$ but not $\lambda_m <_n \lambda_{m+1}$. Now first note that all adjacent labels are labels of the same (process) type (this does not imply that *all* labels are of one and the same type, because quadruple labels have two types). We consider the case that λ_m is simple, say of type i, and λ_{m+1} is quadruple, say $\lambda_{m+1} = \langle v, c, l, m \rangle$ with l of type i (the other three cases are similar). Then it follows that $\lambda_m <_i \langle v, c, l, ? \rangle$ and not $\lambda_m <'_i \langle v, c, l, ? \rangle$, so $h \lceil_h i \neq h' \lceil_h j$.

□

Finally, we are ready for the main theorem of this Section. Let $PR = [P_1 :: S_1 \parallel \ldots \parallel P_n :: S_n]$.

Theorem 9.6 *The proof system presented in this paper is complete respectively K-complete, i.e.*

1. *if* $\models \{\varphi_i\}S_i\{\psi_i\}$ *then* $\vdash \{\varphi_i\}S_i\{\psi_i\}$
2. *if* $\models \{\varphi\}PR\{K_G\psi\}$ *then* $\vdash \{\varphi\}PR\{K_G\psi\}$ $(G \subseteq \{1, \ldots, n\})$

The second clause of this theorem asserts that any correctness formula involving a postcondition referring to the knowledge of (groups of) processes can be derived in our axiom system, provided it is valid.

Proof 1. to prove $\vdash \{\varphi_i\}S_i\{sp(\varphi_i, S_i)\}$. The proof proceeds by induction on S_i:

- skip:

  $\vdash \{(\exists hx[\varphi_i[hx/hist] \wedge hist = hx \cdot l])[hist \cdot l/hist]\}l : skip$
  $\{\exists hx[\varphi_i[hx/hist] \wedge hist = hx \cdot l]\}$ (axiom 7)
  $\vdash \{(\exists hx[\varphi_i[hx/hist] \wedge hist \cdot l = hx \cdot l])\}l : skip$
  $\{\exists hx[\varphi_i[hx/hist] \wedge hist = hx \cdot l]\}$
  $\vdash \{\varphi_i\}l : skip\{\exists hx[\varphi_i[hx/hist] \wedge hist = hx \cdot l]\}$ (rule of conseq.)

- assignment:

  $\vdash \{(\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge x = e[x'/x] \wedge hist = hx \cdot l])[hist \cdot l/hist, e/x]\}$
  $l : x := e$
  $\{\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge x = e[x'/x] \wedge hist = hx \cdot l]\}$ (axiom 8)
  $\vdash \{(\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge e = e[x'/x] \wedge hist \cdot l = hx \cdot l])\}l : x := e$
  $\{\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge x = e[x'/x] \wedge hist = hx \cdot l]\}$
  $\vdash \{\varphi_i\}l : x := e\{\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge x = e[x'/x] \wedge hist = hx \cdot l]\}$ (cons.)

- output:

  $\vdash \{(\exists hx[\varphi_i[hx/hist] \wedge hist = hx \cdot \langle e, c, l, ?\rangle])[hist \cdot \langle e, c, l, ?\rangle/hist]\}l : c!e$
  $\{\exists hx[\varphi_i[hx/hist] \wedge hist = hx \cdot \langle e, c, l, ?\rangle]\}$ (axiom 9)
  $\vdash \{(\exists hx[\varphi_i[hx/hist] \wedge hist \cdot \langle e, c, l, ?\rangle = hx \cdot \langle e, c, l, ?\rangle])\}l : c!e$
  $\{\exists hx[\varphi_i[hx/hist] \wedge hist = hx \cdot \langle e, c, l, ?\rangle]\}$
  $\vdash \{\varphi_i\}l : c!e\{\exists hx[\varphi_i[hx/hist] \wedge hist = hx \cdot \langle e, c, l, ?\rangle]\}$ (rule of conseq.)

- input:

  $\vdash \{\forall v[(\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge hist = hx \cdot \langle x, c, ?, l\rangle])[v/x,$
  $hist \cdot \langle v, c, ?, l\rangle/hist]]\}l : c?x$
  $\{\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge hist = hx \cdot \langle x, c, ?, l\rangle]\}$ (axiom 10)
  $\vdash \{\forall v[(\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge hist \cdot \langle v, c, ?, l\rangle = hx \cdot \langle v, c, ?, l\rangle])]\}$
  $l : c?x\{\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge hist = hx \cdot \langle x, c, ?, l\rangle]\}$
  $\vdash \{\varphi_i\}l : c?x\{\exists x', hx[\varphi_i[x'/x, hx/hist] \wedge hist = hx \cdot \langle x, c, ?, l\rangle]\}$ (conseq.)

- sequential composition:

  By induction hypothesis, $\vdash \{\varphi_i\}S_1\{sp(\varphi_i, S_1)\}$ and $\vdash \{sp(\varphi_i, S_1)\}S_2$ $\{sp(sp(\varphi_i, S_1), S_2)\}$. Then, by rule 3: $\vdash \{\varphi_i\}S_1; S_2\{sp(sp(\varphi_i, S_1), S_2)\}$, or, equivalently, $\vdash \{\varphi_i\}S_1; S_2\{sp(\varphi_i, S_1; S_2)\}$.

- guarded statement:

  By induction hypothesis, for $1 \leq j \leq m$ we have $\vdash \{\varphi_i \wedge b_j\}l_j : c_j?x_j; S_j$ $\{sp(\varphi_i \wedge b_j, l_j : c_j?x_j; S_j)\}$. Applying the consequence rule, we get for $1 \leq j \leq m$ $\vdash \{\varphi_i \wedge b_j\}l_j : c_j?x_j; S_j\{\bigvee_{j=1}^{m} sp(\varphi_i \wedge b_j, l_j : c_j?x_j; S_j)\}$ Hence, using rule 7 we obtain $\vdash \{\varphi_i\}[]_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{(\varphi \wedge \bigwedge_{j=1}^{m} \neg b_j) \vee sp(\varphi_i \wedge b_j, l_j : c_j?x_j; S_j)\}$, or, equivalently, $\vdash \{\varphi_i\}[]_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{sp(\varphi_i, []_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j])\}$.

- iterated guarded statement:

  Let $\star[]_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]$ be given. In a standard way (see e.g. [Hoo93]) we can show the existence of an invariant I for which the following hold:

  1. $I \in Assn_i$
  2. $\models sp(\varphi_i, \star[]_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]) \leftrightarrow (I \wedge \bigwedge_{j=1}^{m} \neg b_j)$
  3. $\models \varphi_i \rightarrow I$
  4. $\models \{I \wedge \bigvee_{j=1}^{m} b_j\}[]_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{I\}$

  Essentially, I expresses the strongest invariant for the statement considered. Now using induction hypothesis and (d) we obtain $\vdash \{I \wedge \bigvee_{j=1}^{m} b_j\}[]_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{I\}$. Using rule 8 we get $\vdash \{I\} \star []_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{I \wedge \bigwedge_{j=1}^{m} \neg b_j\}$. Using (b),(c) and the consequence rule we arrive at $\vdash \{\varphi_i\} \star []_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j]\{sp(\varphi_i, \star[]_{j=1}^{m}[b_j; l_j : c_j?x_j \rightarrow S_j])\}$.

2. Let $G = \{1, ..., n\}$.

Suppose $\models \{\varphi\}PR\{K_G\psi\}$. Let, for all i, $\overline{x_i}$ denote the list of i-local variables in VAR. Define $\overline{\varphi} = \varphi[\overline{v_1}/\overline{x_1}, ...\overline{v_n}/\overline{x_n}, hx/hist] \wedge \overline{x_1} = \overline{v_1} \wedge ... \wedge \overline{x_n} = \overline{v_n} \wedge hist$ $= hx$ ($hx, \overline{v_i}$ fresh for all i).
Clearly, $\overline{\varphi} \rightarrow \varphi$, so $\models \{\overline{\varphi}\}PR\{K_G\psi\}$ holds.
Now let $\varphi_i = loc_i(\overline{\varphi})$, where loc_i is defined below.
Then $\overline{\varphi} \leftrightarrow \bigwedge_i \varphi_i$. Now by definition of $sp(\varphi_i, S_i)$, we have $\models \{\varphi_i\}S_i\{sp(\varphi_i, S_i)\}$.

Thus, by 1. it follows $\vdash \{\varphi_i\}S_i\{sp(\varphi_i, S_i)\}$.
By rule 8 it follows that $\vdash \{\varphi_i\}P_i :: S_i\{K_i(sp(\varphi_i, S_i))\}$, all i
By K-persistence, $\vdash \{\varphi_i\}PR\{K_i(sp(\varphi_i, S_i))\}$, all i
Then, by conjunction, $\vdash \{\bigwedge_i \varphi_i\}PR\{\bigwedge_i K_i(sp(\varphi_i, S_i))\}$
Group knowledge: $\vdash \{\bigwedge_i \varphi_i\}PR\{\bigwedge_i K_G(sp(\varphi_i, S_i))\}$
Distrib. of K_G over $\wedge$ (lemma 7.1): $\vdash \{\bigwedge_i \varphi_i\}PR\{K_G \bigwedge_i(sp(\varphi_i, S_i))\}$
K-axiom, and $\bigwedge_i(sp(\varphi_i, S_i)) \rightarrow \psi$ (see below): $\vdash \{\bigwedge_i \varphi_i\}PR\{K_G\psi\}$
Consequence: $\vdash \{\overline{\varphi}\}PR\{K_G\psi\}$
Variable substitution rule: $\vdash \{\varphi\}PR\{K_G\psi\}$

There remain the definition of loc_i and the proof of $\models \bigwedge_i(sp(\varphi_i, S_i)) \rightarrow \psi$.

For any i, the function $loc_i : Assn_- \rightarrow Assn_i$ is defined as follows:

- $loc_i(\varphi_i) = \varphi_i$
- $loc_i(\varphi_j) = \surd \ (i \neq j)$
- $loc_i(e_1 = e_2) = e_1 = e_2$ if $\mathsf{VAR}(e_1, e_2) \bigcap \mathsf{VAR}_j = \emptyset$
  $= \surd$ otherwise
- $loc_i(\varphi_- \wedge \varphi'_-) = loc_i(\varphi_-) \wedge' loc_i(\varphi'_-)$

where $\wedge'$ is defined as $\wedge$ except for $\surd \wedge' \varphi = \varphi \wedge' \surd = \varphi$.

In this definition, the symbol $\surd$ expresses the projection of a non-i-formula, i.e. a part of the source formula which is of no importance to the formula φ_i. This also explains its role with respect to the operator $\wedge'$: this operator just forgets operands of type $\surd$, rendering the output of loc_i into the domain $Assn_i$. Note that by construction of $\overline{\varphi}$, there always is at least one sub-formula of $\overline{\varphi}$ that is left unprocessed by loc_i (namely $hx \restriction i = hist \restriction i$), so that we need not define $\surd \wedge' \surd$; we can eliminate the $\surd$'s by associating in the right way.

Lastly, suppose $\langle\sigma, h\rangle \models \bigwedge_i sp(\varphi_i, S_i)$. Then, by lemma 6.2 it follows that, for all i, $\langle\sigma, h\rangle \restriction i \models_i sp(\varphi_i, S_i)$. So, by definition of $sp(\varphi_i, S_i)$, for all i:

$\exists\langle\sigma_0^i, h_0^i\rangle[\langle\sigma_0^i, h_0^i\rangle \models_i \varphi_i \wedge \langle\sigma, h\rangle \restriction i \in [\![S_i]\!](\langle\sigma_0^i, h_0^i\rangle)]$.

Now because execution of S_i does not influence the values of logical variables, we have $\sigma \restriction i(x) = \sigma_0^i(x)$ for all $x \in \mathsf{LVAR}$, and $\sigma \restriction i(hx) = \sigma_0^i(hx)$ for all $hx \in \mathsf{LHVAR}$. Hence we can construct σ_0 such that $\sigma_0 \restriction i = \sigma_0^i$ for all i (take $\sigma_0(x_i) = \sigma_0^i(x_i)$ and $\sigma_0(hx) = \sigma(hx)$).

Furthermore, because we have $\langle\sigma_0^i, h_0^i\rangle \models hx \restriction i = hist \restriction i$ (all i) it follows that $\sigma_0^i(hx) = h_0^i$, hence $\sigma \restriction i(hx) = h_0^i$ (all i) and hence, by lemma 9.5, there is some h_0 uniquely determined by $\sigma(hx) = h_0$, such that $h_0 \restriction i = h_0^i$ for all i.

Therefore we can 'melt' together all $\langle\sigma_0^i, h_0^i\rangle$ to obtain a global point $\langle\sigma_0, h_0\rangle$ such that for all i: $\langle\sigma_0, h_0\rangle \restriction i = \langle\sigma_0^i, h_0^i\rangle$.

It follows that $\langle\sigma_0, h_0\rangle \models \bigwedge_i \varphi_i$, hence $\langle\sigma_0, h_0\rangle \models \overline{\varphi}$.

So, we have

$\exists\langle\sigma_0, h_0\rangle[\langle\sigma_0, h_0\rangle \models \overline{\varphi} \wedge \forall i[\langle\sigma, h\rangle \restriction i \in [\![S_i]\!](\langle\sigma_0, h_0\rangle \restriction i)]]$

$\Leftrightarrow \exists\langle\sigma_0, h_0\rangle[\langle\sigma_0, h_0\rangle \models \overline{\varphi} \wedge \langle\sigma, h\rangle \in POSS(P_1, \ldots, P_n, \langle\sigma_0, h_0\rangle)]$

It then follows from $\models \{\overline{\varphi}\}PR\{K_G\psi\}$ that $POSS(P_1, ..., P_n, \langle\sigma_0, h_0\rangle) \models \psi$, hence $\langle\sigma, h\rangle \models \psi$, which proves $\models \bigwedge_i(sp(\varphi_i, S_i)) \to \psi$.

□

10 Some Remarks on the Semantics

The semantics of programs as defined in Section 3 is limited in a certain way. This is because only *after* execution of some process, something about its knowledge can be stated. As a result, group knowledge of any group of processors only exists a posteriori.

Therefore, it would be nice to be able to describe the evolution of knowledge *during* computation of a process. Of course, our Hoare logic would have to be extended in order to be able to reason about these intermediate states, but this can be done for instance by adding an invariant to our triples, in a similar way as used in Pandya's I-logic [Pan88].

In the following however, we will show that it is not at all straightforward to define such a semantics.

First of all, note that an equivalence relation R_i on $\mathcal{S} \times \mathcal{H}$ defines a partition of $\mathcal{S} \times \mathcal{H}$ and vice versa (for the moment, we are not interested in maximal parallelism; therefore we simplify the framework somewhat):

$$\mathcal{S} \times \mathcal{H}/_{R_i} = \{[\langle\sigma, h\rangle]_{R_i} \mid \langle\sigma, h\rangle \in \mathcal{S} \times \mathcal{H}\}$$

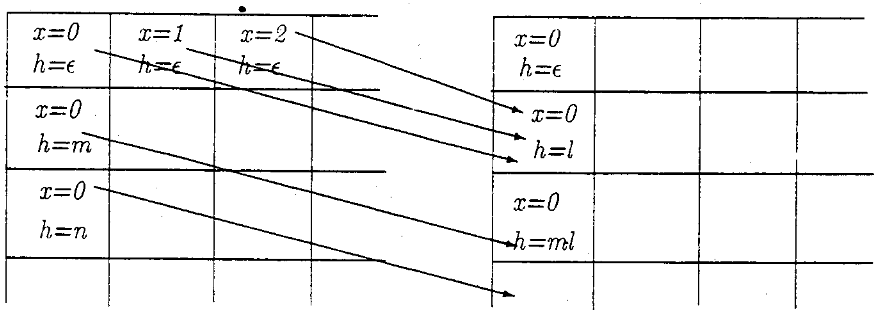
Let us denote the elements of this partition by V_{R_i}, V'_{R_i} etc..

The global idea is now to define the local semantics in terms of transitions of the relation R_i; in other words, when executing some statement S, the new relation R'_i is obtained from the current R_i by determining the semantical image of each class of the partition:

$$\mathcal{M}(S)(\mathcal{S} \times \mathcal{H}/_{R_i}) = \{[\![S]\!](V_{R_i}) \mid V_{R_i} \in \mathcal{S} \times \mathcal{H}/_{R_i}\}$$

As we will explain below, the right hand side of this "definition" does not represent a partition of $\mathcal{S} \times \mathcal{H}$.

Assume a process with only one local variable, x. In the pictures below, each equivalence class is denoted by stating the value of x and the value of the local history, denoted by h. Such a class then consists of all global extensions (with respect to the full program under consideration) of these values.

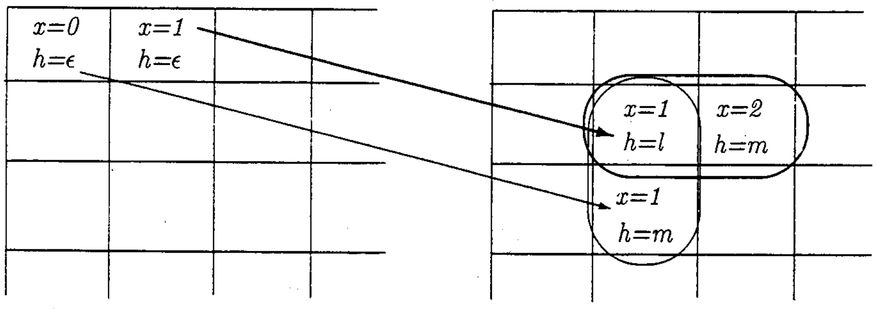


In the first picture, it is shown how the execution of a local command $l : x := 0$ affects the partition. In particular, it is clear that the resulting set of image-classes does not cover the whole of $\mathcal{S} \times \mathcal{H}$, i.e. is not a partition (for instance, all classes that have $h = \epsilon$ do not have an original). Luckily, this is not too serious a bug: we can fix it by defining a "rest" class V_{rest}, as follows:

$$V_{\text{rest}} = \mathcal{S} \times \mathcal{H} \backslash \{[\![S]\!](V_{R_i}) \mid V_{R_i} \in \mathcal{S} \times \mathcal{H}/_{R_i}\}$$

This way, by adding V_{rest} to the image, we again get a partition. Initially we set $V_{\text{rest}} = \emptyset$; note that in the construction of the new partition, V_{rest} does not play any role; it is determined by the images of the "proper" classes.

A second, more serious problem concerns the fact that the set of classes resulting from the execution of a particular statement need not be disjoint. An example of this type of event is represented in the second picture below.



The picture shows the transition of two classes under the semantics of the statement $[\![l : x := 1 [\!] m : x := x + 1]\!]$. Obviously, both resulting equivalence classes are not disjoint, but they do not coincide, either. This type of "clash" occurs because of the use of a nondeterministic choice in combination with smartly chosen assignments. It shows that there exist circumstances in which the resulting partition is such that one would like to consider two points both R_i-equivalent and *not* R_i-equivalent at the same time.

For this second flaw we do not have a satisfying solution as yet; for instance taking as a resulting class the union of the two classes above leads to classes that are too big, unnecessarily diminishing what can be known by the process.

It would be worthwhile investigating this issue further, because there seems not to be an obvious solution to the problem of defining an "update" semantics. For instance, it may be possible to define such a semantics in the case of some not-so-distinguishing view functions.

11 Asynchronous Communication

In this section, we study the modification of the proof system presented in this paper to asynchronous communication. This communication will be modeled by infinite FIFO (first-in-first-out) buffers. Hence, the sending of a value can always take place, while receiving a message requires that the associated buffer be non-empty.

Basically, when shifting from synchronous communication (handshake) to asynchronous communication, the description of the communication interface between processes by means of communication histories becomes more complex in the sense that we now have to represent a successful communication between two processes by two records—one representing the sending action and one representing the receiving action—and, moreover, the sending of a message has to be done first.

An example may clarify things: consider the program

$$[l_1 : c??x; l_2 : d!!3 \parallel m_1 : c!!5; m_2 : d??y]$$

in which two processes communicate asynchronously (indicated by !! and ??) over the channels c and d. The communication behaviour of the individual processes can then be described by the following two sequences of records, where we only record the local label of the sender/receiver): $\langle c??, v, l_1\rangle \cdot \langle d!!, 3, l_2\rangle$ and $\langle c!!, 5, m_1\rangle \cdot \langle d??, v', m_2\rangle$. On the communication history of the combined process, we have to impose the restriction that the projection on each individual process respects its (local) behaviour, as is the case with synchronous communication. However, this is not enough, as it would allow for the history $\langle c!!, 5, m_1\rangle \cdot \langle c??, v, l_1\rangle \cdot \langle d??, v', m_2\rangle \cdot \langle d!!, 3, l_2\rangle$ which is clearly not a valid one, because the value over channel d is earlier received than it is sent. (Synchronously speaking, there would be no problem whatsoever, because communication is now considered as handshake, leaving as single possibility the global history $\langle c, 5, m_1, l_1\rangle \cdot \langle d, 3, l_2, m_2\rangle$). Thus, in defining the semantics of the parallel composition, apart from the projection property, we have to add a constraint with respect to the order in which messages are sent and received.

In the proof system we propose in this section, we localise this constraint by defining 'local' merge predicates M_i (which are in fact global predicates as we shall see) stating that on all in-going channels from process i, at all times enough values have been produced by its environment to enable i's execution. Conjoining these local merge predicates can then easily be seen to coincide with Pandya's NETINV predicate [Pa].

We proceed with discussing the successive adaptations of the framework.

The Language: we replace the communication commands $c?x$ and $c!e$ by $c??x$ resp. $c!!e$ to denote the asynchronous nature of the communication.

Semantics: with respect to the domain, the set of local labels is now defined as

$$\Lambda_i = \{l, \langle v, c??, l\rangle, \langle v, c!!, l\rangle \mid l \text{ appears syntactically in } S_i\}.$$

Here $\langle v, c??, l\rangle$ will denote the event of an input of value v from channel c, where the executed action is labeled with l, and $\langle v, c!!, l\rangle$ likewise for output. By the fact that communication actions are now autonomous, the label does not contain unknown sender/receiver labels as in the the synchronous case.

We define $\mathcal{H}_i(\ni h)$ to be the set of posets (sequences) over Λ_i, for $i \in \mathcal{P}$.

Global semantics: We define $\mathcal{S}$, with typical element σ as the set of global states.

Let $\Lambda = \bigcup_{i \in \mathcal{P}} \Lambda_i$, and define $\mathcal{H}$ to be the set of posets over Λ.

We assume a restriction operator $\lceil$, which can be used in two ways. Firstly it yields a local point when provided with a global point and a process index:

$$\lceil : \mathcal{S} \times \mathcal{H} \times \mathcal{P} \to \bigcup_{i \in \mathcal{P}} (\mathcal{S}_i \times \mathcal{H}_i)$$

by the intuitively obvious restriction/projection operation. Secondly, when given a directed input/output channel as second argument, it yields a sub-trace of a global history consisting of those labels that involve the specified channel. e.g.

$$\langle 3, c!!, l \rangle \cdot m \cdot \langle 3, c??, n \rangle \lceil c?? = \langle 3, c??, n \rangle$$

In the definitions below, we will have need for a prefix operator $\leq$ on posets. Intuitively, $h_1 \leq h_2$ means that h_1 is a prefix of h_2. This notion of prefix is not so easily defined as in the case of linear structures such as streams. The reason for this is that a poset can be extended in more directions.

Definition 11.1 Let $h_1 = (H_1, <_1), h_2 = (H_2, <_2)$, and let λ, μ range over $H_1 \cup H_2$. Then we define

$$h_1 \leq h_2 \Leftrightarrow H_1 \subseteq H_2 \;\wedge\; <_1 = <_2 \cap H_1 \times H_1 \;\wedge\; \forall \lambda \in H_2 \backslash H_1 \;\neg\exists \mu \in H_1[\lambda <_2 \mu]$$

The first two conjuncts express that h_1 is a sub-poset of h_2. The third conjunct states that h_2 is a proper 'right-extension' of h_1. In other words: if some label λ is in $H_2 \backslash H_1$, then also μ is in $H_2 \backslash H_1$, for all μ with $\lambda <_2 \mu$.

Now, we will define the chaotic closure operator CC, which yields all global points that are possible extensions of a given set of local points. Unlike in the synchronous case, this definition will not be just the inverse projection, but an additional conjunct is required in order to guarantee that every receive action is preceded by a corresponding send action. We define this predicate as follows.

Definition 11.2 The predicate $Merge_i$ is defined by

$$(\mathcal{S} \times \mathcal{H} \ni)\langle \sigma, h \rangle \models Merge_i \iff \bigwedge_{c \in IN(S_i)} \forall h' \leq h[h' \lceil c?? \leq h' \lceil c!!]$$

Definition 11.3 The chaotic closure operators $CC' : \mathcal{S}_i \times \mathcal{H}_i \to \mathcal{P}(\mathcal{S} \times \mathcal{H})$ and $CC : \mathcal{P}(\mathcal{S}_i \times \mathcal{H}_i) \to \mathcal{P}(\mathcal{S} \times \mathcal{H})$ are defined as follows:

$$CC'(\langle \sigma, h \rangle) = \{\langle \sigma', h' \rangle \mid \langle \sigma', h' \rangle \lceil i = \langle \sigma, h \rangle \wedge \langle o', h' \rangle \models Merge_i\}$$

$$CC(H) = \bigcup_{\langle \sigma, h \rangle \in H} CC'(\langle \sigma, h \rangle)$$

Given these definitions, we can now fix the relation R_{V_i} for any set of local points V_i:

$$\langle\sigma, h\rangle R_{V_i} \langle\sigma', h'\rangle \iff [\langle\sigma, h\rangle \in maxpar(CC(V_i)) \Leftrightarrow \langle\sigma', h'\rangle \in maxpar(CC(V_i))]$$

So again, as in the synchronous case, the set $\mathcal{S} \times \mathcal{H}$ is divided into precisely two classes for each process.

Syntax of Formulae The syntax of our assertion language is the same as in the synchronous case, except for the addition of the special predicate Merge_i, reflecting the semantical $Merge_i$:

$$Assn_{-} \quad \varphi_{-} :: \quad e = e' \mid \varphi_i \mid \varphi_{-} \wedge \varphi'_{-} \mid \mathsf{Merge}_i$$

Thus, it now becomes possible to infer non-trivial knowledge about non-local information, as we will see in the proof system below. Note that this was not possible in the synchronous case.

Semantics of Formulae The function $\mathcal{T}_{-}$ is extended by the following clause:

$$\mathcal{T}_{-}(\mathsf{Merge}_i)(\langle\sigma, h\rangle) = \bigwedge\nolimits_{c \in IN(S_i)} \forall h' \leq h[h' \restriction c?? \leq h \restriction c!!]$$

Proof System The only different rule is that of knowledge introduction. Because the chaotic closure operator is more selective than in the synchronous case through the enforced validity of $Merge_i$, we can infer more knowledge of the process involved (if process i does some input, it knows some matching output must have occurred beforehand (although i does not know the corresponding send-label)).

Rule 12 *(K-introduction)*

$$\frac{\{\varphi_i\}S_i\{\varphi'_i\}}{\{\varphi_i\}P_i :: S_i\{K_i(\varphi'_i \wedge Merge_i)\}}$$

Completeness

The key lemma is now the following, stating that a global history is completely determined by all its projections, if the $Merge_i$ predicate holds for each i-projection. This lemma enables us to use the abbreviation also in the new situation.

Lemma 11.4 *Let $h, h' \in \mathcal{H}$. Suppose there are $h_i \in \mathcal{H}_i$ such that for all i with $1 \leq i \leq n$ it holds that $h \restriction i = h' \restriction i = h_i$. Suppose furthermore that $h \models \bigwedge_i Merge_i$ and $h' \models \bigwedge_i Merge_i$, and that both h and h' are maximal parallel under these conditions. Then it follows that $h = h'$.*

12 Conclusion

In this paper we have presented a proof system for the correctness of a simple parallel programming language using a logic in which epistemic operators are included to be able to speak about the knowledge of the sub-processes involved in the execution of parallel programs in this language. As we have seen this proof system comprises of a local and a global part. The former is classical dealing with the correctness of local processes, whereas the latter part concerns the parallel composition of processes and eventually the whole process of parallel computation.

It is in this latter part where the use of epistemic operators comes into the picture. These operators enable us to (still) refer to assertions along with the agents (processes) that know them. Combining this knowledge to knowledge of larger groups of processes eventually gives us the desired assertions known by the process as a whole but again we still can refer to the knowledge of every subgroup of processes when we want to. So combining knowledge into group knowledge does not destroy the information about what is known by subgroups. Moreover, this knowledge can also be re-used when putting subgroups in another context (another program), thus supporting the re-use of software and specifications. This illustrates the modularity of our approach. So. summarizing, one could state that in our proof system, the constructs on the local level are handled in a more or less standard way, whereas the parallel (top) construct is treated by means of epistemic operators.

This two-leveledness in our approach is not strictly needed. With suitable adaptations of the semantics it should be possible to allow an arbitrary depth of nesting of the parallel construct.

The section on asynchronous communication shows that the essential factor in devising a proof system as defined in this paper entails the determination of the right view function; this suggests that the view function is a parameter in the proof system that can be tuned so as to match particular cases of interest.

There have been other attempts at defining and proving the notion of knowledge in distributed systems, of which we mention [KT86]; they used an interleaving semantics, as opposed to our poset semantics (a form of true concurrency semantics, cf. [BRR89]), and, moreover, their proof method is based on the well-known proof systems of [AFdR80] and [OG76], and is therefore not compositional.

We believe that the epistemic approach to the correctness of parallel programs may be used fruitfully for a range of programming languages. In particular, since our approach is agent-oriented we believe that the approach is amenable to object-oriented parallel programming languages since the objects in these languages are exactly the agents/processes involved in the execution of a program. Also the incorporation of parallel languages into more advanced agent-oriented software systems as proposed in the realm of distributed AI ([BG88]) might be facilitated in this way. This will be investigated in future research.

We would finally like to mention that we do not need a merging lemma ([Apt83]), due to compositionality of the semantics (cf. also [AdB94]).

acknowledgement Jozef Hooman is acknowledged for providing his useful lecture notes, and for suggesting the use of channeled communication in earlier versions of this paper ([HM94]).

References

[AdB94] P.H.M. America and F.S. de Boer. Reasoning about dynamically evolving process structures. *Formal Aspects of Computing*, 6:269–316, 1994.

[AFdR80] K.R. Apt, N. Francez, and W.-P. de Roever. A proof system for communicating sequential processes. *ACM-TOPLAS*, 2(3):359–385, 1980.

[Ame89] P.H.M. America. Issues in the design of a parallel object-oriented language. *Formal Aspects of Computing*, 1(4):366–411, 1989.

[Apt83] K.R. Apt. Formal justification of a proof system for communicating sequential processes. *Journal of the ACM*, 30:197–216, 1983.

[BG88] A.H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*, San Mateo, CA, 1988. Morgan Kaufmann.

[BRR89] J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*. Springer-Verlag, 1989. Lecture Notes in Computer Science 354.

[dB80] J. W. de Bakker. *Mathematical Theory of Program Correctness*. Prentice-Hall, 1980.

[FLP84] N. Francez, D. Lehmann, and A. Pnueli. A linear-history semantics for languages for distributed programming. *Theoretical Computer Science*, 32:25–46, 1984.

[Gol87] Robert Goldblatt. *Logics of Time and Computation*, volume 7 of *CSLI Lecture Notes*. Stanford, 1987.

[HC84] G.E. Hughes and M.J. Cresswell. *A Companion to Modal Logic*. Methuen, 1984.

[HHM93] W. van der Hoek, M. van Hulst, and J.-J.Ch. Meyer. Towards an epistemic approach to reasoning about concurrent programs. In G. Rozenberg J.W. de Bakker, W.-P. de Roever, editor, *Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*, pages 261–287. Springer-Verlag, 1993.

[HM85] J.Y. Halpern and Y.O. Moses. A guide to the modal logics of knowledge and belief. In *Proc. 9th IJCAI*, pages 480–490, 1985.

[HM90] J.Y. Halpern and Y.O. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[HM94] M. van Hulst and J.-J.Ch. Meyer. An epistemic proof system for parallel processes. In R. Fagin, editor, *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the fifth conference (TARK 1994)*, pages 243–254. Morgan Kaufmann, 1994.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hoo93] J. Hooman. Verification of parallel systems, 1993. Course Notes.

[HZ87] J.Y. Halpern and L.D. Zuck. A little knowledge goes a long way: Simple knowledge-based derivations and correctness proofs for a family of protocols. In *Proc. of 6th PODC*, pages 269–280, 1987.

[Krö87] F. Kröger. *Temporal Logic of Programs*. Springer, 1987.

[KT86] S. Katz and G. Taubenfeld. What processes know: definitions and proof methods. *ACM-PODC*, pages 249–262, 1986.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[OG76] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976.

[Pan88] P.K. Pandya. *Compositional Verification of Distributed Programs*. PhD thesis, Tata Institute of Fundamental Research, Homi Bhabha Road, Bombay 400 005, INDIA, 1988.

[vdHvLM95] W. van der Hoek, B. van Linder, and J.-J.Ch. Meyer. Group knowledge isn't always distributed (nor is it always implicit). In M. Koppel and E. Shamir, editors, *Proc. BISFAI'95*, pages 191–200, Ramat Gan, Jerusalem, 1995.

[Zwi88] J. Zwiers. *Compositionality, Concurrency and Partial Correctness*. PhD thesis, Technical University Eindhoven, 1988.